Jarmo Pihlajaniemi

REST-pohjaisen web-rajapinnan kehittäminen

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Tietotekniikan koulutusohjelma
Insinöörityö
3.5.2012

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Jarmo Pihlajaniemi REST-pohjaisen web-rajapinnan kehittäminen 38 sivua 3.5.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmistotekniikka
Ohjaaja(t)	web-suunnittelija Matias Mäki yliopettaja Erja Nikunen

Tässä työssä perehdyttiin REST-arkkitehtuurityyliin ja sen soveltamismahdollisuuksiin http-protokollaa hyödyntävissä web-rajapinnoissa. Käytännön vertailukohteena toimi Taloussanomien iOS-sovelluksen käyttöön tarkoitettu web-rajapinta. Työssä selvitettiin, kuinka hyvin Taloussanomien nykyinen web-rajapinta vastaa REST-arkkitehtuurityyliä ja mitä asioita pitäisi huomioida rajapinnan kehityksessä. Samalla Taloussanomien web-rajapintaan toteutettiin tuki maksulliselle uutispalvelulle.

REST-käsitteen merkityksestä vallitsee yleinen epätietoisuus. Yleinen käsitys tuntuu olevan, että REST liittyy jotenkin web-rajapintoihin ja että se on kovassa nosteessa nykyaikana. Todellisuudessa REST on kuitenkin yleinen ohjelmistoarkkitehtuurityyli, jota on sovellettu Webin kehityksessä jo sen alkuajoilta asti. Periaatteessa kaiken webin sisällön tulisi noudattaa REST-periaatteita, myös web-rajapintojen. Käytännössä REST-periaatteista on kuitenkin luistettu useissa tapauksissa.

Tämän työn tarkoituksena oli hahmottaa tarkemmin REST-käsitteen merkitystä erityisesti web-rajapintakehityksessä. Erityisenä kiinnostuksen kohteena oli selvittää, missä tapauksissa on oleellista noudattaa REST-periaatteita. Työssä käytiin läpi vallitsevia käsityksiä erilaisista tavoista toteuttaa web-rajapintoja ja selvitettiin, miten nämä eri tavat sijoittuvat suhteessa REST:iin.

Tutkimustyön tuloksena selvisi, että REST-arkkitehtuurityyli tarjoaa monia etuja web-rajapintojen kehityksessä ja sitä kannattaa yleensä noudattaa ainakin jossain määrin. REST-periaatteita kunnioittaen on mahdollista toteuttaa monipuolisia web-rajapintoja, mutta tämä vaatii usein tarkkaa aiheeseen perehtymistä ja ajattelumallin muuttamista palvelukeskeisistä arkkitehtuureista resurssikeskeiseen arkkitehtuuriin. Usein paras vaihtoehto on jonkinlainen kompromissi, jossa otetaan käyttöön REST:n hyvät ominaisuudet niiltä osin, kuin ne on saavutettavissa ilman liikaa vaivannäköä. Tämä toteutui Taloussanomien web-rajapinnan kehityksessä.

Avainsanat	web-rajapinnat, web services, REST, SOAP, RPC, HTTP, URI

Metropolia

Abstract

Author Title Number of Pages Date	Jarmo Pihlajaniemi Developing REST based web API 38 pages 3 May 2012
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Matias Mäki, Web Designer Erja Nikunen, Principal Lecturer

The main goal of this Thesis was to gain more knowledge on Representational State Transfer (REST) based web API design. This knowledge was then used to evaluate to what extent the Taloussanomat Web API follows the REST architectural style and whether there is some ways to improve the RESTfulness of the API. Taloussanomat Web API was built for Taloussanomat iOS application to serve as a gateway to news articles and customer databases.

The major part of this Thesis concentrates on defining the main concepts behind the REST architectural style and web APIs. After getting a good grip on REST I proceed on studying the Hypertext Transfer Protocol (HTTP) and Uniform Resource Identifiers (URI), the two specifications that define the generic interface used by all component interactions on the Web. After getting the ground work done I was able to define Web APIs as “APIs for web based machine-to-machine interaction” as opposed to Web APIs used in human-to-machine interactions, which contains all the web applications used by web browsers.

After clearing up the basic vocabulary involved in Web API design I started to gather information on different approaches to Web API design and was able to narrow Web APIs in two basic categories: REST and Remote Procedure Call (RPC). As a rough characterization REST is the preferred way of developing Web APIs as it's the architectural style specifically designed for the web whereas RPC is a way to abstract remote communication to work like local communication by forgetting and not dealing with all the pitfalls of remote communication.

Finally I go through the Taloussanomat Web API in detail and resolve that although some of the principles of REST based Web APIs have been ignored in developing the Taloussanomat API it is still fairly RESTful and the compromises done where justified in order to produce a practical Web API for its purpose.

Keywords	Web API, Web Services, REST, SOAP, RPC, HTTP, URI

Sisällys

Lyhenteitä ja käsitteitä

HTTP	Hypertext Transfer Protocol. Webissä sijaitsevien komponenttien välistä kommunikaatiota ohjaava protokolla.
Hypermedia	Useasta toisiinsa linkitetystä mediayksiköstä koostuva media.
Esitys	Webissä sijaitsevan resurssin tilaa kuvaava esitys.
Rajoite	Arkkitehtuurityyliin liitetty piirre, jonka tarkoitus on täyttää arkkitehtuurityylille asetetut vaatimukset.
REST	Representational State Transfer. Sovellusarkkitehtuurityyli hajautetuille hypermediajärjestelmille. Ohjaa webin kehitystä.
Resurssi	Mikä tahansa webissä sijaitseva kohde, joka voidaan nimetä yksilöllisesti. Resurssi voi olla esimerkiksi dokumentti, kuva tai hetkellinen palvelu.
RPC	*Remote Procedure Call*. hajautetuille järjestelmille tarkoitettu sovellusarkkitehtuurityyli, joka pyrkii abstraktoimaan etäkutsut lokaaleiksi kutsuiksi.
SOAP	XML-RPC:n pohjalle rakennettu monipuolinen viestintäprotokolla.
URI	*Unifrom Resource Identifier*. Resurssin yksilöivä tunniste.
web	HTTP-protokollan avulla keskenään kommunikoivista komponenteista koostuva maailmanlaajuinen tietojärjestelmä.
web-rajapinta	Webissä sijaitseva rajapinta palvelimen resursseihin.
Web Service	SOAP-protokollaa hyödyntävä web-rajapinta.

XML	Extensible Markup Language. Merkintäkieli, jonka avulla tiedon merkitysvoidaan esittää tiedon yhteydessä.
XML-RPC	XML-merkintäkieltä ja HTTP-protokollaa hyödyntävä RPC-pohjainen viestintäprotokolla hajautetuille järjestelmille.

1 Johdanto

REST-pohjaiset web-rajapinnat ovat olleet viime aikoina kovassa nosteessa [Rodriquez 2008]. Web-rajapintoihin perehtyvä web-kehittäjä törmää helposti juttuihin, joissa REST:n kerrotaan olevan jokin uusi mullistavan tehokas, mutta yksinkertainen tapa toteuttaa web-rajapintoja. REST esitetään yleensä vastakohtana muinaiselle, monimutkaiselle ja raskaalle SOAP:lle, jota käyttävät enää ainoastaan suuret yritykset, joiden on vaikeaa muuntaa jo kehitettyjä SOAP-pohjaisia järjestelmiään REST-pohjaisiksi.

Lähes kaikki tuntuvat olevan yhtä mieltä siitä, että REST on paras vaihtoehto web-rajapinnoille. Jopa monet SOAP:n ja sen pohjana olleiden teknologioiden kehittäjistä ovat viime aikoina, jos eivät suorastaan kääntäneet takkiaan [Vinoski 2009], niin ainakin joutuneet myöntämään tehneensä merkittäviä virheitä toimiessaan REST-periaatteiden vastaisesti [Box 2010]. Tämä ei kuitenkaan tarkoita sitä, että kaikki olisivat REST:stä yhtä mieltä, päinvastoin yksi suurimmista kiistanaiheista web-maailmassa tuntuu tätä nykyä olevan juuri REST-pohjainen web-rajapinta; kysymys ei ole niinkään siitä, onko REST-pohjainen web-rajapinta hyvä, vaan siitä, mikä ylipäätään on REST-pohjainen web-rajapinta ja mikä se ei ainakaan ole.

Täyttä selvyyttä ei itseasiassa ole edes siitä, mikä on web-rajapinta tai itse web, tai edes internet. Aihe vaatii perusteellista selvitystä. Aloitan tutkimusmatkan webin kehityksessä keskeisesti mukana olleen Roy Fieldingin kirjoittamasta väitöskirjasta, jossa hän esittelee REST-käsitteen. Samalla perehdyn webin historiaan ja sen keskeisiin protokolliin HTTP:hen ja URI:in. Tämän jälkeen keksin mielekkään määritelmän web-rajapinnalle ja pyrin erottelemaan erilaisia tapoja toteuttaa web-rajapintoja, joita vertailen REST-pohjaisiin web-rajapintoihin. Lopuksi esittelen Taloussanomien iOS-sovellukselle toteutetun web-rajapinnan, jota kehitin samaan aikaan, kun valmistelin tätä insinöörityötä.

Taloussanomat on Suomen suurin talousaiheinen verkkolehti, ja se kuuluu viestintäkonserni Sanomaan, johon kuuluvat myös mm. Helsingin Sanomat ja Ilta-Sanomat. Taloussanomat on maksuton verkkolehti, joka saa tulonsa pääosin mainosmyynnistä. Kesällä 2011 julkaistiin Taloussanomien iOS-sovellus, joka tarjosi

iOS-laitteille yksinkertaisen ja suorituskykyisen käyttöliittymän Taloussanomien uutistenlukuun. Minun työni oli jatkokehittää rajapintaa tukemaan maksullista uutispalvelua, jonka avulla tilaajille voidaan tarjota maksullista uutissisältöä. Taloussanomien web-rajapinnan kehityksessä käytettiin apuna Sanomien yhteistä SNCDS-taustajärjestelmää sekä Applen iOS-sovelluskehitykseen tarjoamaa maksurajapintaa.

Taloussanomien web-rajapinta pyrkii noudattamaan mahdollisuuksien mukaan REST-periaatteita, mutta tärkeimpänä asiana kehityksessä pidettiin kuitenkin käytännöllisyyttä, jolloin REST-periaatteiden tarkkaan noudattamiseen ei kulutettu liikaa voimavaroja. Tässä työssä käyn tarkemmin läpi Taloussanomien web-rajapinnan ja selvitän, miltä osin se noudattaa REST-periaatteita ja missä kohdin on jouduttu tekemään kompromisseja, ja olisiko jotain voitu tehdä toisin.

2 REST-arkkitehtuurityyli

Tässä luvussa kerrotaan yleisesti REST (REpresentational State Transfer) -arkkitehtuurityylistä, joka toimii myös Taloussanomien web-rajapinnan pohjana. Suoraviivaiset REST-pohjaiset rajapinnat kasvattavat jatkuvasti suosiotaan verrattuna monimutkaisena pidettyihin SOAP-pohjaisiin rajapintoihin. REST ei kuitenkaan liity ainoastaan rajapintoihin, vaan se on itse asiassa yleinen arkkitehtuurityyli, jota voi hyödyntää myös muunlaisten sovellusten kehityksessä.

Tämän luvun ensimmäisessä aliluvussa käydään läpi REST-arkkitehtuurityylin kehityshistoriaa ja sen merkittävää roolia koko WWW:n kehityksessä. Toisessa aliluvussa perehdytään REST:n rakenteeseen ja käydään läpi ne keskeiset arkkitehtuuriset rajoitteet, jotka määrittelevät REST-arkkitehtuurityylin. Kolmannessa aliluvussa käsitellään REST:n soveltamista WWW:n keskeisten protokollien HTTP:n ja URI:n kehityksessä.

2.1 REST:n historia

REST:n kehittäjän Roy Fieldingin määritelmän mukaan REST on sovellusarkkitehtuurityyli hajautetuille hypermediajärjestelmille [Fielding 2000, s. 3].

Hypermediajärjestelmä tarkoittaa järjestelmää, joka koostuu toisiinsa linkitetyistä mediadokumenteista. Tunnetuin REST-arkkitehtuurimallia noudattava, hajautettu hypermediajärjestelmä on WWW (World Wide Web). REST kehitettiinkin alun perin nimenomaan WWW:tä varten, mutta sitä voi hyödyntää myös WWW:stä riippumattomien sovellusten kehityksessä aivan kuten mitä tahansa muuta sovellusarkkitehtuurimallia.

Päämotivaationa REST-arkkitehtuurityylin kehitykselle oli tarve hallita web-protokollien (kuten HTTP ja URI) kehitystä. Ensimmäinen versio REST:stä kehitettiin vuosina 1994 ja 1995 samaan aikaan, kun Fielding kumppaneineen kehitti HTTP/1.0-protokollan ja suunnitteli alustavaa versiota HTTP/1.1:stä. REST toimi toisaalta HTTP:n kehityksen ohjenuorana, mutta samanaikaisesti REST:iä itseään kehitettiin HTTP:n kehityksessä ilmenevien tarpeiden pohjalta. REST:iä kutsuttiinkin alunperin nimellä ”HTTP object model”. [Fielding 2000, s. 109.]

Lopullinen nimi Representational State Transfer pyrkii luomaan kuvan siitä, miten hyvin suunnitellun web-sovelluksen tulisi toimia: Web-sovellus on kokoelma toisiinsa linkitettyjä sivuja, jotka kukin esittävät (represent) käyttäjälle yhden tilan (state) sovelluksesta. Tilasta toiseen siirrytään (transfer) sivujen sisältämien linkkien kautta [Fielding 2000, s. 109].

Lopullisen version REST:istä Fielding esittelee väitöskirjassaan vuonna 2000. Väitöskirjassaan Fielding käy läpi useita olemassa olevia arkkitehtuurityylejä sekä esittelee WWW:n alkuvaiheen kehityksessä ilmenneet vaatimukset, jotka johtivat lopulta uuden arkkitehtuurityylin REST:n kehitykseen.

2.2 REST:n rajoitteet

Fielding määrittelee arkkitehtuurityylin hallittujen rajoitteiden kokonaisuudeksi, jolle on annettu nimi. Arkkitehtuurityylin sisältämien rajoitteiden tarkoituksena on mahdollistaa arkkitehtuurille asetettujen vaatimusten täyttyminen [Fielding 2000, s. 13]. Tässä luvussa esitellään REST-arkkitehtuurimallin sisältämät rajoitteet, joiden tarkoituksena on täyttää WWW:n kehitykselle aikoinaan asetetut vaatimukset. WWW:n kehityksessä ilmenneet päävaatimukset uudelle arkkitehtuurityylille olivat skaalautuvuus,

rajapintojen yleisluontoisuus, komponenttien itsenäisyys sekä mahdollisuus käyttää välikomponentteja vähentämään komponenttien välisen kanssakäymisen viivettä, parantamaan tietoturvaa sekä kapseloimaan vanhentuneita järjestelmän osia. [Fielding 2000, s. 75.]

Näiden vaatimusten täyttämiseksi REST:iin otettiin mukaan rajoitteita useista muista tietoverkkopohjaisista arkkitehtuurityyleistä. REST:n päärajoitteita ovat asiakas-palvelin (client-server), tilaton (stateless), välimuisti (cache), yhdenmukainen rajapinta (uniform interface), kerroksittainen järjestelmä (layered system) sekä ladattava koodi (code-on-demand) [Fielding 2000, s. 96 - 103]. Keskeisin REST:n muista (tietoverkkoihin liittyvistä) arkkitehtuurityyleistä erottava rajoite on yhdenmukainen rajapinta [Fielding 2000, s. 81].

2.2.1 Asiakas-palvelin

Asiakas-palvelin-rajoitteen pääajatus on jakaa sovelluksen toiminnallisuus kahteen alueeseen, joista toinen on asiakkaan ja toinen palvelimen vastuulla. Yleinen tapa jakaa vastuut on laittaa asiakas huolehtimaan käyttöliittymästä samalla, kun palvelin huolehtii tiedon tallennuksesta. [Fielding 2000, s. 45.] Esimerkiksi WWW-järjestelmässä asiakkaan tehtävää hoitaa yleensä www-selain, joka tarjoaa graafisen käyttöliittymän www-palvelinten tarjoamiin palveluihin. Yleinen www-palvelimen tarjoama palvelu on palvelimella sijaitsevaan tietokantaan tallennetun tiedon haku ja muokkaus.

Vastuunjakaminen mahdollistaa yksinkertaisemmat komponentit, mikä parantaa järjestelmän skaalautuvuutta. Lisäksi vastuunjako mahdollistaa asiakkaan ja palvelimen toisistaan riippumattoman kehittämisen, kunhan asiakkaan ja palvelimen välinen rajapinta säilyy muuttumattomana. [Fielding 2000, s. 46.]

2.2.2 Tilaton

Tilattomuus-rajoite on asiakas-palvelin-rajoitteen lisärajoite. Tilattomuus edellyttää, että asiakas-palvelin-session tilaa ei tallenneta palvelimelle. Jokainen asiakkaan tekemä pyyntö on itsenäinen kokonaisuus, joka sisältää kaiken tiedon, jonka palvelin tarvitsee pyynnön käsittelyssä. Session tila tallennetaan siis kokonaisuudessaan asiakkaalle. [Fielding 2000, s. 47.]

REST:n kannalta oleellinen etu tilattomuudesta on skaalautuvuuden parantuminen: koska palvelimen ei tarvitse tallentaa pyyntöjen välistä tilaa, palvelimen rakenne yksinkertaistuu entisestään, ja lisäksi palvelin voi tarvittaessa nopeasti vapauttaa resursseja. Muita tilattomuuden etuja ovat järjestelmän näkyvyyden ja luotettavuuden parantuminen. Näkyvyyden parantumisella tarkoitetaan järjestelmän monitoroinnin helpottumista, kun asiakkaan tekemän pyynnön todellinen luonne on suoraan nähtävissä kussakin pyynnössä. Luotettavuus puolestaan parantuu, koska tilaton järjestelmä kykenee palautumaan vikatilanteista nopeammin. [Fielding 2000, s. 47.]

Tilattoman järjestelmän haittana on verkon mahdollinen kuormittuminen, koska mitään asiakkaan ja palvelimen välistä kontekstia ei voi tallentaa palvelimelle vaan kaikki oleellinen tieto täytyy välittää joka pyynnön yhteydessä. Lisäksi sovelluksen tilan tallentaminen asiakaspäähän vähentää palvelimen kontrollia sovelluksen käyttäytymisestä: sovelluksen oikeanlainen käyttäytyminen tulee riippuvaiseksi asiakkaan tavasta soveltaa sovittua semantiikkaa. [Fielding 2000, s. 79.]

2.2.3 Välimuisti

Tilattomuus-rajoitteesta aiheutuvien verkon suorituskykyongelmien kompensoimiseksi REST:iin on lisätty välimuisti-rajoite. Välimuisti-rajoite edellyttää, että palvelimen vastauksessa asiakkaan pyyntöön on oltava mukana tieto siitä, voidaanko vastauksen sisältämä data tallentaa välimuistiin. Jos välimuistiin tallentaminen on sallittu, asiakas voi tallentaa pyynnön datan omaan välimuistiinsa ja hakea sen tarvittaessa sieltä sen sijaan, että tekisi samanlaisen pyynnön uudestaan palvelimelle. [Fielding 2000, s. 79.]

Välimuisti-rajoitteen ansiosta jotkin pyynnöt voidaan siis jättää kokonaan tai osittain tekemättä, mikä parantaa suorituskyvyn lisäksi, skaalautuvuutta ja käyttäjäkokemusta vähentämällä toimintojen keskimääräistä viivettä. Välimuisti-rajoitteen haittapuolena on luotettavuuden heikkeneminen, sillä välimuistissa voi olla tallennettuna vanhentunutta dataa. [Fielding 2000, s. 80.]

2.2.4 Yhdenmukainen rajapinta

Yhdenmukainen rajapinta järjestelmän komponenttien välillä yksinkertaistaa järjestelmän kokonaisarkkitehtuuria ja parantaa komponenttien välisen kanssakäymisen näkyvyyttä. Yhdenmukainen rajapinta myös mahdollistaa komponenttien itsenäisen kehityksen asiakas-palvelin-järjestelmässä.

Yhdenmukaisen rajapinnan haittapuolena on suorituskyvyn heikkeneminen, koska komponenttien välillä kulkeva data kulkee aina samassa muodossa riippumatta siitä, mikä olisi paras muoto kullekin sovellukselle.

REST tarkentaa yhdenmukaisen rajapinnan rajoitetta vielä neljällä lisärajoitteella: resurssien tunnistaminen, resurssien manipulointi esitysten (representations) kautta, itseselitteiset viestit sekä hypermedian käyttö sovelluksen tilakoneena. [Fielding 2000, s. 82.]

Resurssilla tarkoitetaan REST:ssä mitä tahansa ”käsitettä”, joka voidaan nimetä yksilöllisesti. Resurssi voi olla esimerkiksi dokumentti, kuva tai hetkellinen palvelu (kuten päivän sää Helsingissä). Resurssi voi myös koostua muista resursseista. Kaikki hypermediajärjestelmän sisäiset viittaukset kohdistuvat johonkin resurssiin. Resurssilla on yksiselitteinen tunniste sekä esitys (representation). [Fielding 2000, s. 88.] Resurssin esitys koostuu datasta ja metadatasta. Metadata koostuu avain-arvo-pareista, joissa avain kuvaa standardia, joka määrittelee arvon rakenteen ja semantiikan. [Fielding 2000, s. 91.] Resurssin esitys voi muuttua ajan mukana (esim. päivän sää), vaikka itse resurssi pysyisi samana.

Resurssien käsittelyyn on standardoitu joukko metodeja, joiden avulla resursseja voidaan muokata. Resursseille on myös olemassa standardoitu joukko mediatyyppejä, jotka kuvaavat resurssin esityksen formaattia, kuten esimerkiksi teksti tai jpeg-kuva.

Viestien itseselitteisyys tarkoittaa, että kaikki viestin käsittelyyn tarvittava tieto sisältyy itse viestiin. Viestien itseselitteisyys mahdollistaa välikomponenttien käytön viestien prosessoinnissa. Viestien itseselitteisyys on seurausta myös aiemmin mainitusta tilattomuus-rajoitteesta, joka edellyttää itseselitteisyyttä sillä perusteella, että

palvelimelle ei voi tallentaa mitään asiakas-palvelin-sessioon liittyvää dataa. [Fielding 2000, s. 91]

2.2.5 Kerroksittainen järjestelmä

Kerroksittainen järjestelmä koostuu kerroksista, jotka kommunikoivat keskenään. Kerrokset on järjestetty hierarkkisesti, niin että kukin kerros voi kommunikoida vain välittömästi ylä- ja alapuolellaan olevien kerrosten kanssa. Asiakkaan ja palvelimen välissä sijaitsevien kerrosten avulla voidaan kapseloida vanhentuneita järjestelmän osia – niin asiakas- kuin palvelinpäässä: vanhentuneen palvelun päälle voidaan rakentaa uudistettu rajapintakerros ja uusia palveluita voidaan suojella vanhentuneilta asiakas-sovelluksilta. Myös järjestelmien tietoturvaa voidaan parantaa välikerrosten avulla, esimerkiksi yritysverkon rajalle voidaan asettaa palomuuri vahtimaan liikennettä muun internetin välillä. [Fielding 2000, s. 83.]

Kerroksittaisen järjestelmän selkein haittapuoli on datan prosessoinnin lisääntymisen aiheuttama viiveen kasvu. Kerroksittaisen järjestelmän tarjoamat mahdollisuudet välimuistin käyttöön kuitenkin tasoittavat merkittävästi prosessoinnista aiheutunutta viiveen kasvua.

Yhdenmukaisen rajapinnan rajoitteen yhteydessä mainittu viestien itseselitteisyys antaa lisäksi välikomponenteille mahdollisuuden muokata viestejä niiden kulkiessa järjestelmän kerrosten lävitse. [Fielding 2000, s. 84.]

2.2.6 Ladattava koodi

Ladattava koodi -rajoite antaa palvelimelle mahdollisuuden laajentaa asiakas-sovelluksen toiminnallisuutta lähettämällä vastauksen mukana koodia, joka ajetaan asiakaspäässä. Tämä helpottaa ja yksinkertaistaa asiakas-sovellusten kehittämistä vähentämällä asiakaspäässä etukäteen toteutettavan toiminnallisuuden määrää. Ladattava koodi kuitenkin heikentää järjestelmän näkyvyyttä ja tämän johdosta ladattava koodi -rajoite on REST:issä vapaaehtoinen. [Fielding 2000, s. 84.]

Vapaaehtoinen rajoite voi kuulostaa ristiriitaiselta käsitteeltä. WWW:n kaltaisessa äärettömän suuressa järjestelmässä (jota varten REST on kehitetty) tällaiselle käsitteelle on kuitenkin paikkansa: vapaaehtoisen rajoitteen vaikutukset ovat voimassa niissä järjestelmän osissa, joissa se on käytössä. Esimerkiksi yrityksen sisäverkossa saatetaan sallia ladattava koodi ainoastaan yrityksen omilta palvelimilta, jolloin muulle maailmalle yrityksen asiakas-päätteet näyttäytyvät asiakkaina, jotka eivät käytä ladattava koodi -rajoitetta. [Fielding 2000, s. 84.]

2.3 REST:n soveltaminen HTTP- ja URI-protokollien kehityksessä

HTTP- (Hypertext Transfer Protocol) ja URI (Uniform Resource Identifier) -protokollat määrittelevät geneerisen rajapinnan kaikelle Webin komponenttien väliselle kommunikaatiolle [Fielding 107]. Fielding oli mukana myös näiden protokollien kehityksessä ja pääarkkitehtinä HTTP/1.1:n kehityksessä.

REST toimi toisaalta ohjenuorana HTTP- ja URI-protokollien kehityksessä, mutta samaan aikaan REST:iä itseään kehitettiin näiden protokollien kehityksessä ilmenneiden tarpeiden pohjalta. HTTP-spesifikaatio julkaistiin jo vuonna 1997 ja URI-spesifikaatio vuonna 1998, mutta lopullinen versio REST:stä esiteltiin vasta vuonna 2000. HTTP:n ja URI:n kehitys tosin jatkuu yhä edelleen.

REST:n tavoin myös URI- ja HTTP-protokollia sovellettiin jo näiden kehitysvaiheessa. Ensimmäinen HTTP/1.1-protokollaa hyödyntävä www-palvelin oli Apache, jonka kehityksessä Fielding oli myös oleellisesti mukana. Fielding vastasi apachen ydinfunktioista, jotka parsivat HTTP-viestejä ja pystyi näin varmistamaan, että Apache toimisi tarkasti HTTP/1.1-protokollan mukaan. HTTP/1.1-protokollan soveltaminen Apacheen vaikutti vastavuoroisesti protokollan itsensä kehitykseen ja näin Apachesta (ja muista varhaisista HTTP/1.1-protokollaa tukevista sovelluksista) saatu käytännön kokemus oli merkittävässä roolissa REST:n kehityksessä. [Fielding 2000, s.137.]

3 HTTP-protokolla

Hypertext Transfer Protocol (HTTP) on protokolla, joka määrittelee sovelluskerroksen keinot tiedonsiirtoon hypermediajärjestelmässä [RFC 2616 1999, s. 7]. Sovelluskerros

on ylin kerros internetin liikennöintiä kuvaavassa TCP/IP-mallissa, jonka muita kerroksia ovat kuljetuskerros, verkkokerros ja linkkikerros. Asiakkaan ja palvelimen väliset viestit muunnetaan sovellusmuodosta kuljetusmuodon ja verkkomuodon kautta linkkimuotoon ja jälleen takaisin sovellusmuotoon kunkin pyynnön ja vastauksen yhteydessä. Sovelluskerros on sovelluskehityksen kannalta oleellisin kerros. [TCP/IP model 2012.]

HTTP-protokolla noudattaa REST-arkkitehtuurityyliä, eli se toteuttaa REST:n asettamat rajoitteet. HTTP määrittelee yhdenmukaisen rajapinnan (yhdenmukainen rajapinta -rajoite) asiakas-palvelin-järjestelmälle (asiakas-palvelin-rajoite). Järjestelmän tilattomuus (tilattomuus-rajoite) riippuu siitä, tallennetaanko sovelluksen tila palvelimelle, joten http ei siihen suoranaisesti vaikuta. Http:n tarjoama yhdenmukainen rajapinta kuitenkin mahdollistaa itseselitteiset viestit (kaikki viestin käsittelyssä tarvittava tieto sisältyy itse viestiin), mikä on yksi edellytys tilattomuudelle. Itseselitteiset viestit tukevat lisäksi kerroksittaisen järjestelmän rajoitetta.

Http toteuttaa yhdenmukaisen rajapinnan määrittelemällä tarkan rakenteen ja syntaksin viestille, jonka avulla resurssien esityksiä kuljetetaan asiakkaan ja palvelimen välillä. Asiakas voi manipuloida resursseja http-viestin sisältämän resurssin esityksen kautta. Resurssin esitys voi sisältää linkkejä, joiden kautta asiakas voi noutaa uuden resurssin esityksen palvelimelta, eli siirtyä uuteen tilaan sovelluksessa (hypermedia sovelluksen tilakoneena). Http-viesti sisältää resurssin esityksen lisäksi metadataa, jonka avulla palvelin voi esimerkiksi kertoa, saako asiakas tallentaa viestin sisällön välimuistiin (välimuisti-rajoite).

3.1 Resurssien tunnistaminen

Uniform Resource Identifier (URI) on merkkijono, jolla voidaan tunnistaa resurssi. URI:lla tunnistettava resurssi on REST-arkkitehtuurityylin määritelmän mukainen yksilöllinen kohde, esimerkiksi dokumentti, kuva tai palvelu. Resurssin esitys voi muuttua olosuhteiden mukaan, vaikka itse resurssi pysyy samana. Esimerkiksi päivän sään esittävä resurssi on aina merkitykseltään sama, vaikka sen esittämä sää vaihtuukin koko ajan. [RFC 2396 1998, s. 2.]

URI voi tarkoittaa joko resurssin sijaintia (Uniform Resource Locator, URL) tai nimeä (Uniform Resource Name, URN) tai molempia [RFC 2396 1998, s. 3]. HTTP:n yhteydessä on totuttu käyttämään URL-termiä, sillä HTTP:n tarkoituksena on nimenomaan paikantaa (locate) resurssit tietoverkossa, eikä niinkään nimetä. [RFC 2616 1999, s. 19.]

URI-protokolla määrittelee resurssin tunnisteelle yleisen syntaksin. URI:n yleinen syntaksi on muotoa:

```
<scheme>://<authority><path>?<query>
```

Scheme-komponentti kertoo protokollan, joka määrittelee URI:lle protokollakohtaisen tarkemman syntaksin. Scheme on ainoa pakollinen osa URI:ssa. Muut komponentit ovat vapaaehtoisia, mutta niiden täytyy olla määritellyssä järjestyksessä. Authority-komponentti määrittelee protokollan sisäisen nimiavaruuden tunnisteen loppuosalle. Path-komponentti on varsinainen resurssin tunniste määritellyssä nimiavaruudessa. Query-komponentti sisältää resurssille tulkittavaksi tarkoitettua informaatiota, joka voi esimerkiksi vaikuttaa resurssin esitystapaan. [RFC 2396 1998, s. 12-15.]

URI:a hyödyntävät muut protokollat voivat tarkentaa yleistä syntaksia vastaamaan paremmin omia tarpeitaan. HTTP urlin määrittelemä URI-syntaksi on muotoa:

```
"http:" "//" host [ ":" port ] [ abs_path [ "?" query ]]
```

Alussa oleva ”http:” määrittää käytettävän protokollan. Host tarkoittaa koneen osoitetta, jolla palvelin sijaitsee. Port kertoo isäntäkoneen portin (oletuksena 80), jossa palvelin odottaa asiakkaiden TCP-yhteydenottoa. Abs_path (absoluuttinen polku) on resurssin tunniste palvelimella. Query-komponentti on yleisen URI-spesifikaaiton mukainen, eli tarjoaa informaatiota resurssin tulkittavaksi. Kaikki hakasuluissa olevat kohdat ovat vapaaehtoisia. [RFC 2616 1999, s. 19.]

3.2 Viestien rakenne

HTTP-viesti voi olla asiakkaan pyyntö palvelimelle tai palvelimen vastaus asiakkaan pyyntöön. HTTP-viestit perustuvat vuonna 1982 määriteltyyn yleiseen internetin

viestiformaattiin (Standard for the format of arpa internet text messages). HTTP-viesti koostuu aloitusrivistä, yhdestä tai useammasta ylätunnisterivistä sekä mahdollisesta sisältö-osasta. [RFC 2616 1999, s. 19]

```
GET /rfc/rfc2616 HTTP/1.1
Host: www.ietf.org
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:9.0.1)
Gecko/20100101 Firefox/9.0.1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fi-fi,fi;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
If-Modified-Since: Fri, 28 Jul 2006 14:17:05 GMT
If-None-Match: "9fe0a2-67187-419a4f7871a40"-gzip
Cache-Control: max-age=0
```

Esimerkkikoodi 1. HTTP-pyyntö

Esimerkkikoodi 1 esittää HTTP-pyyntöviestin, jonka tein omalta koneeltani www-selaimella hakeakseni esityksen HTTP-määritelmästä IETF:n palvelimelta. Ylin rivi on aloitusrivi, joka kertoo käytetyn metodin (GET), resurssin sijainnin palvelimella (/rfc/rfc2616) sekä käytetyn HTTP-protokollaversion (HTTP/1.1). Seuraavat rivit ovat viestin ylätunnisteita. Ylätunnisteet kertovat mm. palvelinkoneen osoitteen (host), käytetyn asiakassovelluksen (User-agent). Ylätunnisteiden alle tulisi tyhjällä rivillä erotettuna viestin sisältö, mutta tässä pyynnössä sisältöosaa ei ole.

```
HTTP/1.1 200 OK
Date: Sun, 01 Apr 2012 11:48:12 GMT
Server: Apache/2.2.10 (Linux/SUSE) mod_ssl/2.2.10 OpenSSL/0.9.8h
PHP/5.2.13 with Suhosin-Patch mod_python/3.3.1 Python/2.6
mod_perl/2.0.4 Perl/v5.10.0
Last-Modified: Fri, 28 Jul 2006 14:17:05 GMT
Etag: "9fe0a2-67187-419a4f7871a40"-gzip
Accept-Ranges: bytes
Vary: Accept-Encoding
Content-Encoding: gzip
```

```
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/plain

[tekstimuotoinen esitys resurssista]
```

Esimerkkikoodi 2. HTTP-vastaus

Esimerkkikoodi 2 esittää HTTP-vastausviestin ylempänä esitettyyn esimerkkipyyntöön. HTTP-vastauksessa aloitusrivi alkaa protokollaversiolla (HTTP/1.1), jota seuraa vastauksen statuskoodi ja statusteksti. Aloitusrivin alla on pyynnön tapaan ylätunnistetiedot, jotka kertovat vastauksen tapauksessa mm. viestin luontiajan (Date), palvelinsovelluksen (server) sekä sisällön mediatyypin (Content-Type). Alimmaisena on itse resurssin esitys, joka tässä tapauksessa on ylätunnisteen määritelmän mukaisesti puhdasta tekstiä (Content-Type: text/plain).

HTTP käyttää yleisesti määriteltyjä internetin mediatyyppejä (Internet media type, aiemmin MIME). Mediatyypin tunniste koostuu kahdesta kauttaviivalla erotetusta osasta (tyyppi ja alatyypi) sekä yhdestä tai usemmasta vapaaehtoisesta osasta. text/plain-mediatyypin lisäksi tavallisia mediatyyppejä ovat esimerkiksi text/html, image/png, audio/mpeg. [RFC 2616 1999, s. 26; Internet media type 2012.]

Statuskoodin tarkoituksena on kertoa, kuinka palvelin pääpiirteittäin suoriutui pyynnön toteuttamisesta. Statuskoodia seuraa pyynnössä statusteksti, joka kertoo lyhyesti koodin merkityksen ihmiskäyttäjille. Statuskoodi koostuu kolmesta numerosta, joista ensimmäinen kertoo vastauksen luokan ja kaksi jälkimmäistä erottelevat koodin luokan sisällä. Vastausluokkia on viisi: informatiivinen (1xx), onnistuminen (2xx), uudelleenohjaus (3xx), asiakasvirhe (4xx), palvelinvirhe (5xx).

”Informatiivinen” status tarkoittaa, että palvelin on vastaanottanut viestin ja jatkaa sen käsittelyä. Onnistuminen tarkoittaa, että palvelin onnistui pyynnön suorittamisessa; tällöin vastauksessa on yleensä mukana resurssin esitys. Uudelleenohjaus tarkoittaa, että pyynnön suorittaminen edellyttää toisen resurssin kutsua; tässä tapauksessa vastauksessa on mukana jatkoresurssin tunniste. Asiakasvirhe tarkoittaa, että

pyynnössä on jokin virhe. Palvelinvirhe tarkoittaa, että palvelin epäonnistui pyynnön toteuttamisessa, vaikka itse pyyntö oli virheetön.

Http tukee kahta päätapaa välimuistitukselle: asiakas voi hakea sisällön suoraan omasta välimuistista tai antaa palvelimelle ehdot, joiden täyttyessä pyyntö suoritetaan. Suoraa välimuistitusta varten palvelin asettaa vastauksen ylätunnisteeksi Cache-control tai Expires, jotka määrittelevät, milloin asiakas voi hakea esityksen suoraan välimuistista ja milloin täytyy ottaa yhteys palvelimeen. Expires määrittää ajankohdan (esim. Thu, 01 Dec 1994 16:00:00 GMT), jolloin esitys vanhenee; Cache-controlin avulla on mahdollista määritellä tarkempia ehtoja esityksen vanhenemiselle.

Kun välimuistiin tallennettu esitys vanhenee (expires ja cache-control -kenttien määritelmän perusteella), asiakas pyytää uutta esitystä palvelimelta (tai joltain välikomponentilta). Asiakas voi asettaa pyyntöviestin ylätunnisteeksi ehtoja, kuten If-Modified-Since tai If-Match. If-Modified-Since-kentän arvona on päivämäärä. Jos resurssia on muokattu määritellyn päivämäärän jälkeen, palvelin suorittaa pyynnön. If-Match-kentän arvona on Etag, yksilöllinen tunniste resurssin esitykselle (ei siis itse resurssille, jonka tunnisteena toimii url). Jos resurssin tunniste on eri kuin pyynnössä oleva tunniste, palvelin suorittaa pyynnön.

3.3 Metodit viestien lähettämiseen

REST:n yleisen rajapinnan rajoite edellyttää, että järjestelmällä on joukko standardoituja metodeja resurssien käsittelyyn [Fielding 2000, s. 99]. HTTP määrittelee kahdeksan metodia: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE ja CONNECT.

OPTIONS-metodi palauttaa resurssiin tai palvelimen kykyihin liittyvät mahdollisuudet ja/tai vaatimukset. GET-metodi palauttaa resurssin esityksen. HEAD on muuten sama kuin GET, paitsi että vastauksessa saa olla mukana ainoastaan ylätunnisteet (headers).

PUT-metodilla voidaan luoda uusi resurssi tai päivittää vanhaa: Jos PUT-pyynnön kohde-urlille ei ole määritetty resurssia, luodaan uusi resurssi, jonka tunnisteeksi tulee annettu kohde-url ja sisällöksi PUT-pyynnön sisältö. Jos kohde-url sisältää jo resurssin, päivitetään sen sisältö vastaamaan PUT-pyynnön sisältöä. POST-metodilla voidaan lisätä aliresursseja jo luotuihin resursseihin. Aliresurssi voi olla esimerkiksi

sivuhuomautus dokumenttiin tai viesti keskustelufoorumin viestiketjuun. DELETE-metodilla voidaan poistaa luotu resurssi.

TRACE-metodin on tarkoitus auttaa asiakasta järjestelmän virheentunnistamisessa; TRACE-metodi palauttaa asiakkaan pyynnön sisällön sellaisenaan takaisin palvelimelta asiakkaalle. CONNECT-metodia käytetään sellaisten proxyjen kanssa, jotka voivat dynaamisesti vaihtua tunneleiksi. [RFC 2616 1999, s. 10.] Tämä mahdollistaa asiakkaan ja palvelimen välisen kommunikoinnin käyttäen muuta kuin HTTP-protokollaa, kuten esimerkikis TLS-protokollaa, joka ei salli proxyja [Fielding 2000, s. 96].

HTTP-metodit on jaettu turvallisiin (safe), idempotentteihin (idempotent) ja muihin metodeihin: turvallinen metodi tarkoittaa, että metodilla ei saa olla minkäänlaisia vaikutuksia resursseihin; idempotentti metodi tarkoittaa, että jos metodilla on vaikutus resurssiin, niin usealla peräkkäisellä identtisellä pyynnöllä on sama vaikutus kuin yksittäisellä pyynnöllä. Turvallisia metodeja ovat OPTIONS, GET, HEAD ja TRACE; idempotentteja metodeja ovat turvallisten metodien lisäksi PUT ja DELETE.

4 Web-rajapinnat

WWW:n alkuperäinen tarkoitus oli tarjota palveluita ihmiskäyttäjille: WWW-selainten avulla käyttäjät voivat selata www-palvelimilla olevia dokumentteja (resurssien esityksiä). Www:n käyttötavat kuitenkin monipuolistuivat nopeasti ja pian tuli tarve käsitellä resursseja myös koneellisesti; näin syntyivät web-rajapinnat. Web-rajapinnan tarkoitus on tarjota erilaisille asiakassovelluksille ohjelmointirajapinta www-palvelimilla sijaitsevien resurssien käsittelyyn. Web-rajapintaa käyttävä asiakassovellus voi toimia samanaikaisesti myös itse palvelimena ja tarjota web-rajapintaa muille asiakassovelluksille käyttäen hyväksi yhtä tai useampaa muilla palvelimilla sijaitsevaa web-rajapintaa.

Web-sovelluksen ja web-rajapinnan välinen kommunikaatio ei juurikaan poikkea ihmisen (www-selaimen) ja www-palvelimen välisestä perinteisestä kommunikaatiosta: molemmissa tapauksissa asiakas lähettää HTTP-protokollan mukaisen pyynnön palvelimelle käyttäen URI-protokollan mukaista tunnistetta haluamalleen resurssille.

Ainoa ero on palvelimelta tulevassa palautusviestissä: resurssin esityksestä on karsittu pois kaikki ihmiskäyttäjille tarkoitetut elementit kuten rakenne- (HTML-koodi) ja tyylimäärittelyt (CSS-koodi), jolloin jäljelle on jäänyt pelkkä raaka data, jota on helppo käsitellä koneellisesti.

Web-rajapinta-nimitystä käytetään toisinaan myös ihmiskäyttöön tarkoitetuista web-palveluista; ihmiskäyttäjälle näkyvä graafinen käyttöliittymähän ei tule suoraan sellaisenaan palvelimelta vaan asiakassovellus rakentaa käyttöliittymän palvelimelta saamiensa rakenne- ja tyylitietojen perusteella. Toisaalta vaikka unohdetaan tämä pinnan alla oleva koneellinen rajapinta, ihmiskäyttäjälle näkyvää graafista käyttöliittymää voidaan myös kutsua web-rajapinnaksi. Tässä työssä web-rajapinnalla tarkoitetaan kuitenkin nimenomaan sellaista webin kautta tavoitettavaa rajapintaa, joka tarjoaa tyyli- ja rakennetiedoista riisuttua raakaa dataa koneelliseen käsittelyyn.

Tällaisen web-rajapinnan voi toteuttaa monella tapaa. Vaikka web-rajapinta soveltaa REST-pohjaista HTTP-protokollaa, se ei tarkoita, että itse web-rajapinta olisi varsinaisesti REST-pohjainen. Itseasiassa web-rajapinnat hyvin harvoin noudattavat HTTP-protokollaa (tai yleensäkään REST-periaatteita) täysin täsmällisesti. Usein web-rajapintoja kuitenkin kutsutaan REST-rajapinnoiksi, jos REST-periaatteita on edes pyritty noudattamaan. Eräs suosittu tapa toteuttaa web-rajapintoja poikkeaa kuitenkin niin oleellisesti REST-periaatteista, että tällöin voidaan puhua ihan omasta arkkitehtuurityylistä erotuksena REST-tyylistä. Tätä tyyliä kutsutaan RPC-tyyliksi (Remote Procedure Call).

4.1 Tyypillinen web-rajapinta

Perinteisiin työpöytäsovelluksiin tottunut ohjelmoija mieltää rajapinnan usein joukoksi funktioita, joilla on jokin tehtävä. Funktiolle voidaan antaa tehtävän suorittamisessa tarvittavia parametrejä, ja se voi palauttaa tehtävän tuloksena jonkin arvon, joka kuvaa tehtävän onnistumista tai sitten tehtävän tuloksena luodun muuttujan. Esimerkiksi uutissovelluksen tallennusrajapinta voisi sisältää funktioita, joiden avulla voidaan käsitellä tietokantaan tallennettuja uutisartikkeleita. Rajapinnan sisältämät funktiot voisivat olla esimerkiksi seuraavanlaisia (pseudokielellä esitettynä):

```
createArticle(article);
```

```
getArticle(id);
getArticles(year, month, day);
updateArticle();
deleteArticle(article);
```

Nämä funktiot vastaavat tuttua CRUD-joukkoa (create, read, update, delete), joka muodostaa perustan useimmille tietokantasovelluksille [CRUD 2012]. Funktioilla voidaan luoda, hakea, muokata ja poistaa artikkeleita. Kun tällaisiin rajapintoihin tottunut ohjelmoija laitetaan toteuttamaan sama toiminnallisuus web-rajapintana, tulos näyttää usein tältä:

```
http://www.example.com/createArticle
http://www.example.com/getArticle
http://www.example.com/getArticles
http://www.example.com/updateArticle
http://www.example.com/deleteArticle
```

Eli funktiot on käännetty suoraan web-resursseiksi, joiden urlit muistuttavat funktioiden nimiä. Tarvittavat parametrit annetaan urlin query-komponentissa, esimerkiksi: http://www.example.com/getArticle?id=1234 tai sitten http-viestin sisältönä, jos parametrina on monimutkainen olio, kuten yksittäinen artikkeli updateArticle-funktiossa. Paluuarvot tulevat vastausviestin sisältöosassa.

Asiakassovelluksen päähän on koodattu tallennusmoduuli, joka sisältää edellä kuvatut funktiot. Kun asiakassovellus kutsuu tallennusmoduulin getArticle-funktiota, tallennusmoduuli luo http-pyynnön ja lähettää sen funktiota vastaavaan urliin web-rajapinnassa. Web-rajapinta purkaa http-viestin ja kutsuu vastaavaa funktiota palvelinsovelluksessa. Lopuksi web-rajapinta palauttaa tuloksen http-vastauksena asiakassovellukselle. Muu asiakassovellus käyttää tallennusmoduulia kuin mitä tahansa moduulia tietämättä, että tämä ottaa yhteyden web-rajapintaan.

Tyypillisesti REST-tyyliä noudatetaan vain sen verran kuin on pakko; http-protokollaa noudatetaan vain sen verran kuin www-palvelimet edellyttävät. Esimerkiksi kaikissa kutsuissa on voitu ihan hyvin käyttää pelkkää GET-metodia. Toisaalta, jos GET-metodia on käytetty pelkästään hakutoimenpiteissä ja POST-metodia muissa toimenpiteissä, on metodeja käytetty kuitenkin oikein, niin kuin http-protokolla määrää: GET-metodin

tarkoitus on palauttaa resurssien esityksiä, POST-metodilla voidaan lisätä ja muokata aliresursseja. Tällöin vain jätetään osittain hyödyntämättä http:n tarjoama valmis metodijoukko, jota hyödyntämällä web-rajapinnasta voisi tehdä yhdenmukaisemman webin yleisen arkkitehtuurin kanssa.

4.2 RPC-pohjainen web-rajapinta

Hieman syvällisemmin web-rajapintoihin perehtynyt web-ohjelmoija on saattanut törmätä sellaisiin termeihin kuin RPC (Remote Procedure Call) ja SOAP (Simple Object Access Protocol). Valtaosa merkittävistä web-rajapinnoista käytti näitä teknologioita ennen kuin REST alkoi kasvattaa suosiotaan.

RPC:n eli etäkutsun ideana on kutsua etäyhteyden takana olevaa rajapintaa samalla tavalla kuin lokaalia rajapintaa. Välissä oleva tietoverkkokerros pyritään abstraktoimaan niin, että ohjelmoijan ei tarvitsisi huolehtia siitä vaan hän voisi keskittyä sovelluslogiikan kehittämiseen. RPC:tä voi luonnollisesti soveltaa myös muissa tietoverkoissa kuin webissä. Webin yhteydessä RPC:stä käytetään nimitystä XML-RPC, joka tarkoittaa RPC:tä http:n ja xml:n avulla toteutettuna.

XML-RPC:n idea on sama kuin aiemmin esitetyssä tyypillisen web-rajapinnan tapauksessa, mutta abstraktointiajatus on viety vielä pidemmälle: Kutsuttavien resurssien määrä on supistettu yhteen ja varsinaiset kutsuttavat funktiot on määritelty URI:n sijasta pyyntöviestin sisällössä, joka noudattaa XML-formaattia. Kaikki XML-RPC-pohjaisen web-rajapinnan kutsut kohdistuvat siis ainoastaan yhteen URI:in, esimerkiksi:

```
http://www.example.com/api
```

Koko rajapinta muodostaa siis yhden suuren resurssin, joka parsii pyyntöviestin sisällöstä kutsutun funktion ja parametrit. Käytettävällä http-metodilla ei siis ole mitään merkitystä, kun varsinainen funktio on määritelty viestin sisällössä. Esimerkkisovelluksen artikkelinluonti XML-RPC:llä toteutettuna voisi näyttää tältä (pyyntöviestin sisältö):

```
<?xml version="1.0"?>
```

```
<methodCall>
  <methodName>createArticle</methodName>
  <params>
    <param>
      <name>articleHeader</name>
      <value><string>Artikkelin otsikko</string></value>
    </param>
    <param>
      <name>articleBody</name>
      <value><string>Artikkelin sisältö</string></value>
    </param>
    ...[loput parametrit]...
  </params>
</methodCall>
```

Artikkelinluontipyynnön vastaus palvelimelta koostuisi samantyylisestä xml-koodista, mutta sisältönä olisi määritelty funktion ja parametrin sijasta esimerkiksi status ja mahdollinen virheilmoitus. [XML-RPC 2012.]

SOAP on Microsoftin kehittämä monimutkaisempi versio XML-RPC:stä. Käytännössä SOAP on protokolla, joka määrittelee tarkasti XML-RPC-tyylisessä viestinnässä käytettävän XML-viestin rakenteen [Richardson & Ruby 2007, s.19]. SOAP-viesti koostuu http:n tapaan ylätunnisteista ja sisällöstä, jotka molemmat upotetaan (webin yhteydessä käytettynä) http-viestin sisältöosaan. SOAP:n tärkeimpiä suunnitteluperiaatteita olivat yksinkertaisuus ja laajennettavuus [Don Box et. al. 2000]. Yksinkertaisuus on monen mielestä hyvin kyseenalainen piirre SOAP:n yhteydessä, mutta laajennettavuus sen sijaan on toteutunut hyvin erilaisten SOAP-standardiin liitettyjen ws-standardien myötä (mikä osaltaan on vahvistanut vaikutelmaa SOAP:n monimutkaisuudesta). Ws-standardien tarkoitus on huolehtia erityisesti suuren mittakaavan yritysrajapintoihin liittyvistä ongelmista, kuten transaktioista (WS-AtomicTransaction) ja tietoturvasta (WS-Security). [Spies 2008.]

4.3 REST-pohjainen web-rajapinta

Oleellinen ero RPC-pohjaisen ja REST-pohjaisen web-rajapinnan välillä on se, että RPC-pohjainen web-rajapinta koostuu operaatioista, kun taas REST-pohjainen web-rajapinta

koostuu resursseista [Tilkov 2007]. Periaatteessa kaikki yksilöllisesti tunnistettavat kohteet web-järjestelmässä ovat resursseja, mutta RPC:n tapauksessa resurssit ovat yksinomaan palvelutyyppisiä resursseja, operaatioita. Palvelutyyppiset resurssit eivät palauta varsinaisesti esitystä itsestään vaan jostain toisesta resurssista, jota ei välttämättä edes ole tunnistettu yksilöllisesti urlilla.

REST:n tapauksessa rajapinta koostuu sen sijaan palveluresurssien lisäksi yksinkertaisista resursseista, joihin kohdistetut http-pyynnöt vaikuttavat ainoastaan resurssiin itseensä. Itse asiassa ylivoimaisesti suurin osa REST-pohjaisen web-rajapinnan resursseista on juuri näitä ”yksinkertaisia” resursseja. Esimerkiksi yllä olevan esimerkin mukaisessa uutissovelluksen web-rajapinnassa jokainen artikkeli muodostaisi yksilöllisesti tunnistettavan resurssin.

REST-versio esimerkkisovelluksen web-rajapinnasta näyttäisi tältä:

```
http://www.example.com/articles
http://www.example.com/articles/{year}
http://www.example.com/articles/{year}/{month}
http://www.example.com/articles/{year}/{month}/{day}
http://www.example.com/articles/{year}/{month}/{day}/{id}
```

Jälkimmäisessä urlissa aaltosulkeissa olevat arvot kuvaavat artikkelin luontiaikaa ja id:tä. Id on sovelluksen sisäinen tunniste artikkelille, esimerkiksi tietokantataulussa olevan rivin id. Aaltosulkeiden arvoja muuttamalla saadaan siis jokaiselle artikkelille yksilöllinen tunniste, jolloin nämä ovat REST:n määritelmän mukaisia resursseja. Kun id jätetään pois saadaan url resursseille, jotka koostuvat erilaisista kokoelmista artikkeliresursseja, jotka ovat siis näiden resurssien aliresursseja.

Näiden resurssien avulla voidaan toteuttaa kaikki viisi esimerkissä kuvattua käyttötapausta: luoda artikkeli, hakea artikkeli tai artikkeleita, muokata artikkelia ja poistaa artikkeli. Http-pyynnön seurauksena tehtävä operaatio ei riipu pelkästään urlista vaan pyynnössä käytettävästä http-metodista, joka määritetään pyyntöviestin ensimmäisellä rivillä ennen urlia. Esimerkin käyttötapaukset kuvattuna http-pyynnöillä näyttäisivät siis tältä:

```
POST http://www.example.com/articles
```

```
GET http://www.example.com/articles/{year}/{month}/{day}/{id}
GET http://www.example.com/articles
PUT http://www.example.com/articles/{year}/{month}/{day}/{id}
DELETE http://www.example.com/articles/{year}/{month}/{day}/{id}
```

Articles-resurssi voi joko luoda uuden artikkelin (POST) tai palauttaa kaikki artikkelit (GET). Yksittäisen artikkelin tunnistava resurssi joko palauttaa, muokkaa tai poistaa resurssia vastaavan artikkelin (ja itse resurssin). Määrittämällä esimerkin urleihin päivämäärät, mutta ei id:tä, voitaisiin luoda artikkeleita tietyille päivämäärille tai palauttaa tietyn päivämäärän artikkelit.

REST-pohjainen rajapinta muistuttaa siis eräänlaista hakemistoa, johon voi säilöä uutisartikkeleita. Jokainen päivämäärä muodostaa oman hakemiston, joka koostuu sen päivän artikkeleista. Tällainen hakemistotyyppinen dokumenttienkäsittelyjärjestelmä webin oli tarkoitus alun perin ollakin. Tätä hakemistorakennetta käyttäen on kuitenkin mahdollista toteuttaa rajapinta mille tahansa web-sovelluksille [Tilkov 2007], myös esimerkissä kuvattua monimutkaisemmalle.

Kuinka REST-rajoitteet sitten tarkalleen ottaen toteutuvat edellä kuvatussa rajapinnassa? Mikä tekee edellä kuvatusta rajapinnasta REST-pohjaisen? Asiakas-palvelin-rajoite sisältyy jo webin määritelmään ja on automaattisesti mukana kaikissa web-rajapinnoissa: web koostuu asiakkaista ja palvelimista, jotka kommunikoivat keskenään http-protokollan välityksellä; jos rajapinta ei toteuta näitä piirteitä, se ei ole web-rajapinta.

Ladattavan koodin rajoite on vapaaehtoinen, joten se ei vaikuta rajapinnan REST-pohjaisuuteen. Tilattomuus-rajoite toteutuu, jos sovelluksen tilaa ei yksinkertaisesti tallenneta palvelimelle, eli jokainen pyyntö sisältää kaiken pyynnön toteuttamisessa tarvittavan tiedon.

Välimuisti-rajoite edellyttää, että palvelimen lähettämässä vastausviestissä on mukana tieto siitä, voiko asiakas tallentaa viestin sisältämää dataa välimuistiin [Fielding 2000, s. 79]. Jos rajapinta käyttää hyväksi http-protokollassa määritettyjä keinoja tämän tiedon välittämiseen, välimuisti-rajoite on toteutettu.

Kerroksittainen järjestelmä -rajoite toteutuu noudattamalla http-protokollassa määriteltyä yhdenmukaista rajapintaa: resurssit tunnistetaan URI:lla ja resursseja käsitellään oikeaoppisesti http:n määrittelemien metodien avulla. Hypermedia sovelluksen tilakoneena -rajoite toteutuu, jos resurssien esitykset sisältävät linkkejä sovelluksen seuraaviin mahdollisiin tiloihin, esimerkiksi artikkelilistauksessa on mukana linkit, jokaisen yksittäisen artikkelin esitykseen.

4.4 REST vs. RPC

REST ja RPC voidaan siis käsittää kahdeksi pääsuuntaukseksi web-rajapintojen kehityksessä. Periaatteessa kaikki web-rajapinnat noudattavat REST-periaatteita jossain määrin (web-rajapinta määrittelee vähintään yhden urlilla tunnistettavan resurssin), mutta silloin kun REST-periaatteita on rikottu riittävän paljon, voidaan puhua kokonaan eri tyylistä. Yleensä tämä toinen tyyli muistuttaa silloin enemmän RPC-tyyliä.

Yleinen mielipide tällä hetkellä tuntuu suosivan REST-tyyliä ensisijaisena vaihtoehtona uusien web-rajapintojen kehittämisessä. Mm. CORBA:n (CORBA on eräs RPC:n muoto) kehityksessä vahvasti mukana ollut Steve Vinoski on hyökännyt voimakkaasti RPC-protokollaa vastaan. Vinoskin mukaan RPC:n kehityksessä on keskitytty lähinnä kielen ongelmiin: päätavoitteena on ollut abstraktoida etäkutsut niin, että niitä on voinut käyttää lokaalien kutsujen tapaan. REST:n kehityksessä lähtökohtana on ollut tietoverkon välityksellä tapahtuvan kommunikoinnin haasteet, kun taas RPC:n tapauksessa tietoverkko-ongelmat on pyritty lakaisemaan maton alle [Vinoski 2009].

SOAP:n kehitystä ”mentoroinut” Don Box puolestaan on myöntänyt olleensa ”web-aloittelija” SOAP:n kehityksen aikaan [Box 2010], mikä on oletettavasti ollut suuri syy siihen, miksi http:n tarjoamia etuja ei osattu hyödyntää SOAP:n kehityksessä vaan sitä alettiin rakentaa nimenomaan XML-RPC:n päälle. RPC-tyylin aiheuttamia tietoverkkoihin liittyviä luotettavuusongelmia on pyritty ratkomaan SOAP-viestien sisällä sen sijaan, että olisi hyödynnetty http-viestiä, jonka sisällä SOAP-viestit kulkevat. Tämän seurauksena SOAP-protokolla ja sen lisäosat ovat paisuneet, niin suuriksi ja monimutkaisiksi, että SOAP:n 1.2 määritelmässä on virallisesti luovuttu lyhenteestä Simple Object Access Protocol ja SOAP on nyt vain määritelmän nimi [Rozlog 2010].

Pienimuotoisissa projekteissa, joissa yksinkertaisuus menee luotettavuuden ja skaalautuvuuden edelle, XML-RPC on hyvä vaihtoehto. Valinta XML-RPC:n ja jonkinlaisen hybridi-REST:n välillä riippuu tällöin lähinnä kehittäjän mieltymyksistä. Jos taas tavoitteena on vakavasti otettava rajapinta, jonka kehityksessä luotettavuus on tärkeää, voidaan harkita SOAP-pohjaista rajapintaa tai tiukasti REST-periaatteita noudattavaa rajapintaa [Rozlog 2010].

REST:n etuja ovat karkeasti ottaen suorituskyky ja yksinkertaisuus. SOAP:n etuja taas ovat luotettavuus ja turvallisuus. SOAP häviää REST:lle suorituskyvyssä ja yksinkertaisuudessa nimenomaan laajan ja tarkan määritelmänsä ja sen lisäosien johdosta, jotka kuitenkin vuorostaan tarjoavat SOAP-rajapinnoille luotettavuutta ja turvallisuutta. Taistelu SOAP- ja REST-pohjaisten rajapintojen välillä jatkuu, mutta REST:n yksinkertaisuus, erityisesti pohjautuminen web:n keskeisiin teknologioihin näyttäisi olevan voitollinen suuntaus.

4.5 REST-pohjaisen web-rajapinnan kypsyystasot

Kun mietitään REST:n soveltuvuutta johonkin tiettyyn sovellukseen, kannattaa REST:ä lähteä purkamaan rajoite kerrallaan. Kukin rajoite tarjoaa joitain etuja ja joitain haittoja. Tapauskohtaisesti täytyy päättää, mitkä edut ovat tärkeämpiä kuin niistä seuraavat haitat. Useissa tapauksessa tällaista tarkkaa selvitystä ei edes tehdä, sillä se vie aikaa, ja aika on rahaa, mikä on usein tärkeintä yritysmaailmassa. Käydään nyt kuitenkin läpi joitain REST-arkkitehtuurityylin rajoitteiden etuja ja haittoja web-rajapintoihin sovellettuna.

Asiakas-palvelin-rajoite tulee kuten aiemmin todettiin automaattisesti mukana kaikissa web-rajapinnoissa, joten sen hyötyjä ja haittoja ei ole tarpeen pohtia. Välimuistin käyttöä kannattaa sen sijaan pohtia tarkasti kaikissa vakavasti otettavissa web-sovelluksissa: välimuistin käyttö lisää merkittävästi suorituskykyä, mutta samalla lisää riskiä, että käyttäjälle näytetään vanhentunutta tietoa. Välimuistin käyttö ei ole varsinaisesti rajapinnan piirre vaan yksittäisen resurssin: riippuu resurssin luonteesta, kannattaako sitä välimuistittaa ja miten kauan. Välimuistin käyttö on lisäksi sellainen ominaisuus, jonka käyttöä voidaan säätää muuttamatta varsinaista rajapintaa, resurssien tunnisteita ja kutsutapoja.

Tilattomuus kuuluu myös osittain tähän kategoriaan: Jossain vaiheessa voidaan päättää toteuttaa rajapintaan jokin toiminnallisuus, esimerkiksi transaktio, joka edellyttää sovelluksen tilan tallentamista palvelimelle. Tilattomuus ei kuitenkaan varsinaisesti vaikuta rajapinnan rakenteeseen, yhdenmukaiseen rajapintaan, joka on REST:n tärkein ominaisuus [Fielding 2000, s. 81].

Useimmissa REST-pohjaisia web-rajapintoja käsittelevissä kirjoituksissa unohdetaan kokonaan muut rajoitteet ja keskitytään ainoastaan yhdenmukaisen rajapinnan edellyttämiin rajoitteisiin (mikä on toki luontevaa kun kehityksen alla on nimenomaan rajapinta). Fieldingin määrittelemät yhdenmukaisen rajapinnan alirajoitteet tulkitaan usein hieman toisistaan poikkeavilla tavoilla kirjoittajasta riippuen. Keskeiset REST-pohjaisen web-rajapinnan ominaisuudet, jotka kaikki yleensä hyväksyvät ovat resurssien tunnistaminen URI:lla ja resurssien manipulointi HTTP-metodeilla.

Leonard Richardsonin kehittämässä Richardson Maturity Modelissa (RMM) REST-pohjaiset (tai sellaiseksi pyrkivät) web-rajapinnat on jaettu neljään ”kypsyystasoon”, jotka kuvaavat, miten hyvin web-rajapinta noudattaa REST-periaatteita. Käytännössä tasot kuvaavat REST:n yhdenmukaisen rajapinnan alirajoitteiden toteutusastetta. Tasot on numeroitu nollasta kolmeen. [Richardson 2008.]

Tasolla 0 web-rajapinta ei varsinaisesti toteuta ainuttakaan yhdenmukaisen rajapinnan rajoitetta: rajapinta koostuu käytännössä yhdestä URI:sta ja yhdestä HTTP-metodista, mikä on perusedellytys, jotta rajapintaan voidaan edes ottaa yhteys webin kautta. Useimmat RPC-pohjaiset web-rajapinnat (XML-RPC ja SOAP) ovat tällä tasolla. Eli vaikka muut REST:n rajoitteet olisi toteutettu, ilman yhdenmukaista rajapintaa web-rajapinta olisi edelleen oleellisesti RPC-pohjainen.

Tasolla 1 web-rajapinta on jaettu URI:lla tunnistettaviin resursseihin. Rajapinnan jakaminen erillisiin resursseihin helpottaa rajapinnan rakenteen hahmottamista, mutta resurssit ovat kuitenkin edelleen palvelutyyppisiä. Erona RPC:hen funktionimet on siirretty http-viestin sisällöstä resurssin URI:in, mikä vähentää viestin parsimisen aiheuttamaa kuormaa. Aiemmin esitelty ”tyypillinen web-rajapinta” sopii tälle tasolle. Tällä tasolla on vielä hämärää, muistuttaako rajapinta enemmän RPC- vai REST-pohjaista rajapintaa, käsitelläänkö rajapinnan kautta resursseja vai lähetelläänkö etäkutsuja.

Tasolla 2 web-rajapinta on jaettu URI:lla tunnistettaviin resursseihin, kuten tasolla 1, mutta lisäksi rajapinnassa on otettu käyttöön HTTP-metodit. Yhdenmukaisten metodien käyttö resurssien käsittelyssä tekee rajapinnasta helpommin lähestyttävän ja mahdollistaa välikomponenttien käytön. Tarkkaan määritellyt metodit mahdollistavat lisäksi pyyntöjen optimoinnin metodien ominaispiirteiden mukaisesti [Richardson 2008]. Tällä tasolla resurssit ovat puhtaita resursseja, joihin voi kohdistaa yhdenmukaisen rajapinnan määrittelemiä toimenpiteitä. URI:lla tunnistetaan resurssi - ei resurssien käyttötapaa. Palvelukeskeisestä arkkitehtuurista (RPC) on otettu selkeä askel resurssikeskeiseen arkkitehtuuriin (REST). Tason 2 rajapintoja voi sanoa jo vahvasti REST-pohjaisiksi: harva tämän hetken ”REST-pohjainen” web-rajapinta yltää tälle tasolle.

Tasolla 3 web-rajapinnan REST-pohjaisuus viimeistellään käyttämällä hypermediaa sovelluksen tilakoneena. Tämä tarkoittaa, että asiakkaalle lähetetyissä resurssien esityksissä tulee olla linkki sovelluksen mahdollisiin seuraaviin tiloihin, seuraaviin resurssien esityksiin. Hypermedian luulisi olevan juuri se tekijä, joka erottaa web-selainten käyttämät ”ihmisrajapinnat” koneiden käyttämistä web-rajapinnoista. Kuitenkin monien mielestä hypermedian pitäisi koskettaa myös web-rajapintoja, eli sovelluksen tulisi tietää etukäteen vain yksi url web-rajapintaan, ja urlit jatkotoimenpiteisiin tulisi selvitä vastauksena saadusta esityksestä.

5 Taloussanomien web-rajapinta

Taloussanomat julkaisi kesällä 2011 maksuttoman iOS-sovelluksen, jolla käyttäjät voivat lukea Taloussanomien uutisia iPadille optimoidun käyttöliittymän kautta. Muutamaa kuukautta myöhemmin sovelluksesta julkaistiin 1.1 versio, joka sisälsi mm. iOS5-käyttöjärjestelmää varten tehtyjä parannuksia sekä iPhone-version. iPhone-versio poikkeaa iPad-sovelluksesta pienemmälle näytölle optimoidulla käyttöliittymällä (kuva 1).

Kuvio 1. Vasemmalla näkymä alkuperäisen iPad-sovelluksen käyttöliittymästä ja oikealla sen iPhone-versio.

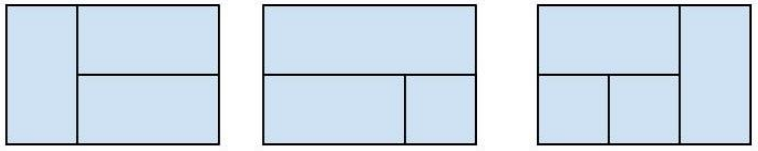
IOS-sovelluksen käyttöliittymä eli asiakassovellus on toteutettu käyttäen hyväksi Applen kehittämää Objective C -pohjaista Cocoa-ohjelmointirajapintaa. Cocoa on tarkoitettu erityisesti Applen Mac OS X- ja iOS-käyttöjärjestelmissä ajettavien sovellusten kehittämiseen. IOS-sovelluksen palvelinpuoli on toteutettu PHP:lla, ja se tarjoaa REST-pohjaisen web-rajapinnan Taloussanomien uutisten hakuun.

Tässä luvussa käsitellään iOS-sovelluksen käyttämää web-rajapintaa: määritellään vaatimukset web-rajapinnalle, käydään läpi web-rajapinnan rakenne ja lopuksi arvioidaan, kuinka hyvin web-rajapinta noudattaa REST-arkkitehtuurityyliä ja missä kohdissa on jouduttu tekemään kompromisseja.

5.1 Web-rajapinnan määrittely

IOS-sovelluksen päätoiminnallisuus on Taloussanomien uutisten lukeminen. Uutiset on jaettu erilaisiin osiohin, kuten markkinointiin, analyyseihin ja kolumneihin. Jokaista osiota varten on oma sivu, jolla listataan ainoastaan osioon liittyviä uutisia. iOS-sovelluksessa näkyvät osiot ovat kokoelmia Taloussanomien web-sivuston osioista.

Etusivulla listataan kaikkien osioiden uutisia. Uutiset listataan joko aikajärjestyksessä tai toimittajien valitsemassa järjestyksessä. Osalla sivuista uutiset listataan yksinkertaisesti alekkain ja osalla uutiset listataan ruudukoittain. Ruudukko on kolme ruutua leveä ja kaksi ruutua korkea, ja se koostuu 1-6 uutisesta, jotka voivat olla 1-3 ruudun kokoisia.



Kuvio 2. Esimerkkejä uutisruudukoista.

Uutista klikkaamalla päästään uutiskohtaiseen näkymään, jossa näytetään yksi uutinen kokonaisuudessaan. Uutiskohtaisessa näkymässä voidaan siirtyä suoraan seuraavaan tai edelliseen uutiseen.

Osa uutisista kuuluu maksulliseen Taloussanomat+-palveluun. Jos käyttäjä ei ole tilannut Taloussanomat+-palvelua, tämä näkee maksulliset uutiset lukon kuvalla varustettuna. Yksittäisen uutisen näkymässä maksullisesta uutisesta näytetään pelkkä ingressi sekä kehotus tilata Taloussanomat+-palvelu. Taloussanomat+-palvelun tilaajat näkevät kaikki uutiset kokonaisuudessaan, ja maksullisten uutisten ohessa on avatun lukon kuva.

Käyttäjä voi tilata Taloussanomat+-palvelun sovelluksen kautta. Tilauksen kestoksi voi valita kuukauden tai vuoden. Tilaukset uusiutuvat automaattisesti tilauskauden päätyttyä, ellei käyttäjä erikseen peru tilausta. Lisäksi käyttäjä voi tehdä ilmaisen koetilauksen, jolla saa Taloussanomat+-palvelun ominaisuudet käyttöönsä kahdeksi viikoksi. Koejakson päättymisen jälkeen käyttäjää muistutetaan koejakson päättymisestä ja kehotetaan tilaamaan maksullinen Taloussanomat+-palvelu. Koejakson voi tilata vain kerran.

Näiden toimintojen toteuttamista varten web-rajapintaan täytyy toteuttaa siis seuraavanlaisia palveluita: Uutislistojen ja uutisruudukoiden haku (nämä järjestetään julkaisuajan tai toimittajien määrittelemän järjestyksen perusteella); yksittäisen uutisen haku kokonaisuudessaan (suorituskyvyn parantamiseksi kokonaisiakin uutisia voi hakea

useamman kerrallaan); tilausten ja koetilausten teko sekä huomautukset koetilauksen päättymisestä.

5.2 Rajapinnan resurssit

Taloussanomien web-rajapinta on toteutettu PHP-ohjelmointikielellä ja se hakee uutiset MySQL-tietokannasta ja käsittelee tilaukset SNCDS-taustajärjestelmän ja Applen tilausrajapinnan kanssa. Rajapinta tarjoilee uutiset asiakkaille JSON-muotoisina olioina, HTTP-viestin sisältönä. Rajapinta pyrkii noudattamaan REST-periaatteita, mutta koska rajapinta on tarkoitettu ainoastaan alihankkijan käyttöön ja aikaa oli rajoitetusti, ei REST-periaatteiden tiukka noudattaminen ollut ykkössijalla rajapinnan suunnittelussa. Taulukossa 1 on esitelty rajapintaan toteutetut resurssit.

Taulukko 1. Taloussanomien web-rajapinnan tarjoamat resurssit

Toiminto	Url	Parametrit (suluissa ei-pakolliset)	Metodi
Uutisten haku	/news/get_news	Uutisen/Uutisten id, (laitteen id)	GET
Uutislistan haku	/news/get_list	(Päivämäärä, Osio)	GET
Uutisruudukon haku	/news/get_grid	(Päivämäärä, Osio)	GET
Tilauksen voimassaolon tarkistus	/orders/is_valid	Laitteen id	GET
Tilaustyyppien haku	/orders/get_types		GET
Tilauksen tallennus	/orders/save	AppStoren kuitti, laitteen id	POST
Koetilauksen tallennus	/customer/save	Laitteen id	POST
Huomautus koetilauksen päättymisestä	/customer/notify	Laitteen id	GET

Alkuperäisessä rajapinnassa oli tarjolla ainoastaan uutistenhakuun tarkoitetut resurssit (kolme ylintä resurssia taulukossa 1). Maksullisen Taloussanomat+-palvelun käyttöönottoa varten toteutettiin loput resurssit.

Uutisosioiden sivuilla ja etusivulla näytetään useampi uutinen kerrallaan, yksinkertaisena listana (/news/get_list) tai ruudukkona (/news/get_list). Molemmille resursseille voidaan antaa parametrina päivämäärä ja osio. Ilman parametreja palautetaan tämän päivän etusivun uutiset, eli sekoitus kaikkien osioiden tärkeimpiä uutisia. Uutisten mukana palautetaan pörssidataa, joka näytetään sovelluksen yläosassa. Uutisruudukon haussa palautetaan uutislistan lisäksi ruudukko-olio, joka määrittelee, miten uutiset sijoittuvat ruudukoihin.

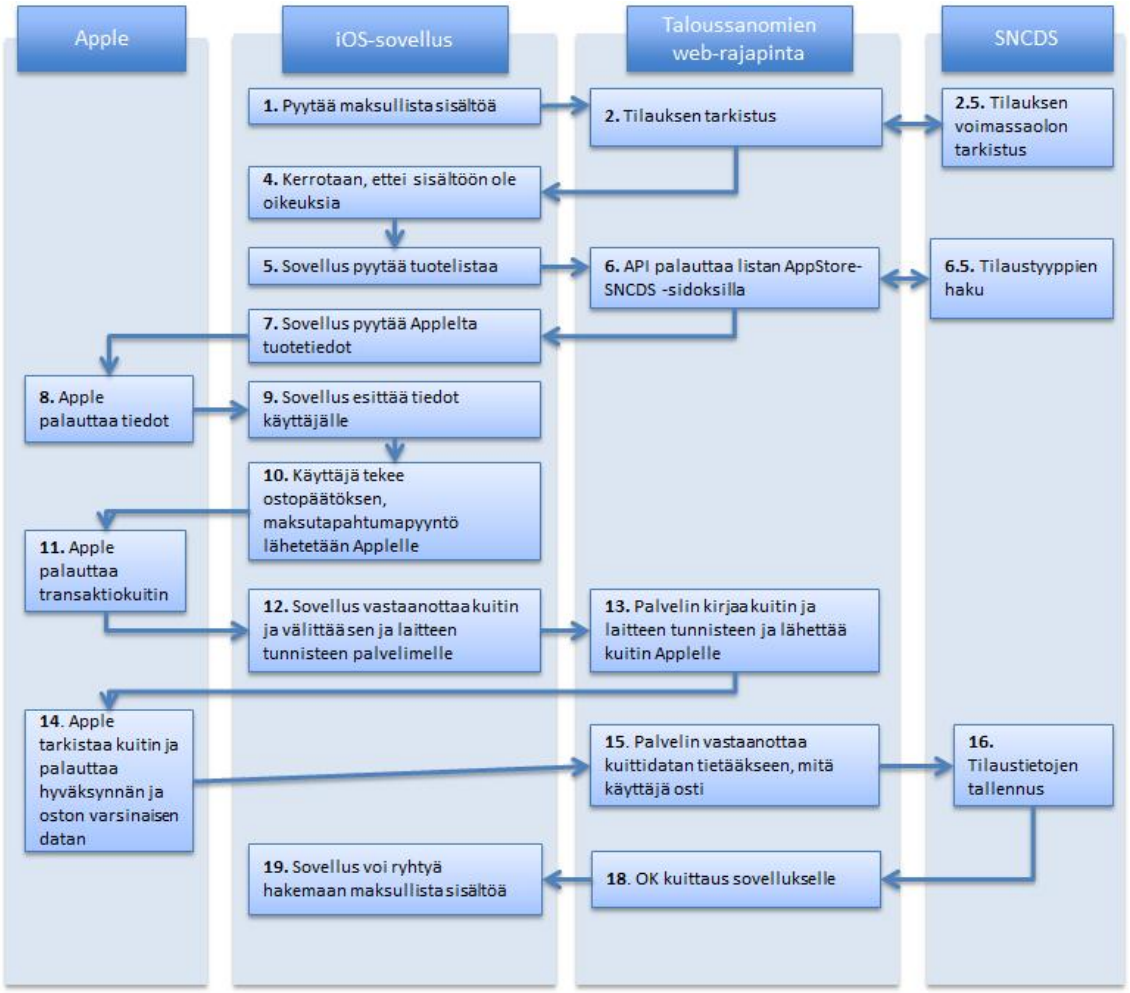
Kokonaisten uutisten hakua (/news/get_news) käytetään uutiskohtaisessa näkymässä, jossa näytetään yksi uutinen kerrallaan, kokonaisuudessaan. Suorituskyvyn parantamiseksi kokonaisiakin uutisia haetaan yleensä useampi kerrallaan ja tallennetaan laitteen välimuistiin; käyttäjän selatessa uutisia uutiskohtaisessa näkymässä sovellus hakee seuraavan uutisen nopeasti välimuistista, mikä parantaa huomattavasti käyttäjäkokemusta. Asiakassovellus asettaa pyynnössä haluamiensa uutisten id:t putkimerkillä (|) eroteltuna yhteen query-parametriin, josta palvelin sitten parsii halutut id:t.

Maksullisia uutisia pyydettäessä asiakas antaa kokonaisten uutisten hakuun lisäparametrina laitteen tunnisteen, jonka perusteella palvelin selvittää onko asiakas tehnyt tilausta. Jos asiakas ei ole tilaaja, maksullisista uutisista palautetaan ainoastaan ingressi; jos asiakas on tilaaja, kokonaisten uutisten lisäksi palautetaan tieto, että asiakkaan tilaus on voimassa. Sovellus tallettaa tämän tiedon sisäiseksi tilakseen, jotta sen ei tarvitse erikseen kutsua tilauksen voimassaolon tarkistus -resurssia (/orders/is_valid). Sovelluksen käyttöliittymä on hieman erilainen riippuen siitä, onko käyttäjä tilaaja vai ei (tilaamista edellyttävät toiminnot näkyvät ei-tilaajalle harmaana yms.).

Tilaustyyppien haku -resurssi (/orders/get_types) palauttaa käytettävissä olevat tilaustyypit (koe-, kuukausi-, ja vuositilaus). Tämän funktion avulla sovellus tarkistaa vastaako sovelluksen käyttämät tilaustyypit varsinaisia SNCDS:ssä olevia

tilaustyyppejä. Jos jostain syystä jotain tilaustyyppiä ei ole tarjolla rajapinnassa, sovellus ei tarjoa sitä myöskään käyttäjälle tilattavaksi.

Tilauksen teossa sovellus käyttää sekä Applen että Taloussanomien rajapintoja. Sovellus hakee tilaustyyppien haku -resurssin palauttamien tuotteiden tiedot (hinta ja kesto) Applen rajapinnasta ja esittää tiedot käyttäjälle. Jos käyttäjä päättää tilata tuotteen, sovellus tekee tilauspyynnön Applen rajapintaan. Applen rajapinta palauttaa onnistuneesta tilauksesta kuitin, jonka sovellus lähettää Taloussanomien Tilauksen tallennus -resurssille (/orders/save), kuitin lisäksi sovellus lähettää laitteen id:n. Taloussanomien rajapinta kutsuu vielä Applen rajapintaa varmistaakseen sovellukselta saamansa kuitin oikeellisuuden. Jos kuitti on aito, Applen rajapinta palauttaa kuittia vastaavat tuotetiedot joiden perusteella rajapinta tallentaa kuukausi- tai vuositilauksen SNCDS-järjestelmään ja palauttaa sovellukselle vastauksen onnistuneesta tilauksesta.



Kuvio 3. Tilauksen teko iOS-sovelluksessa.

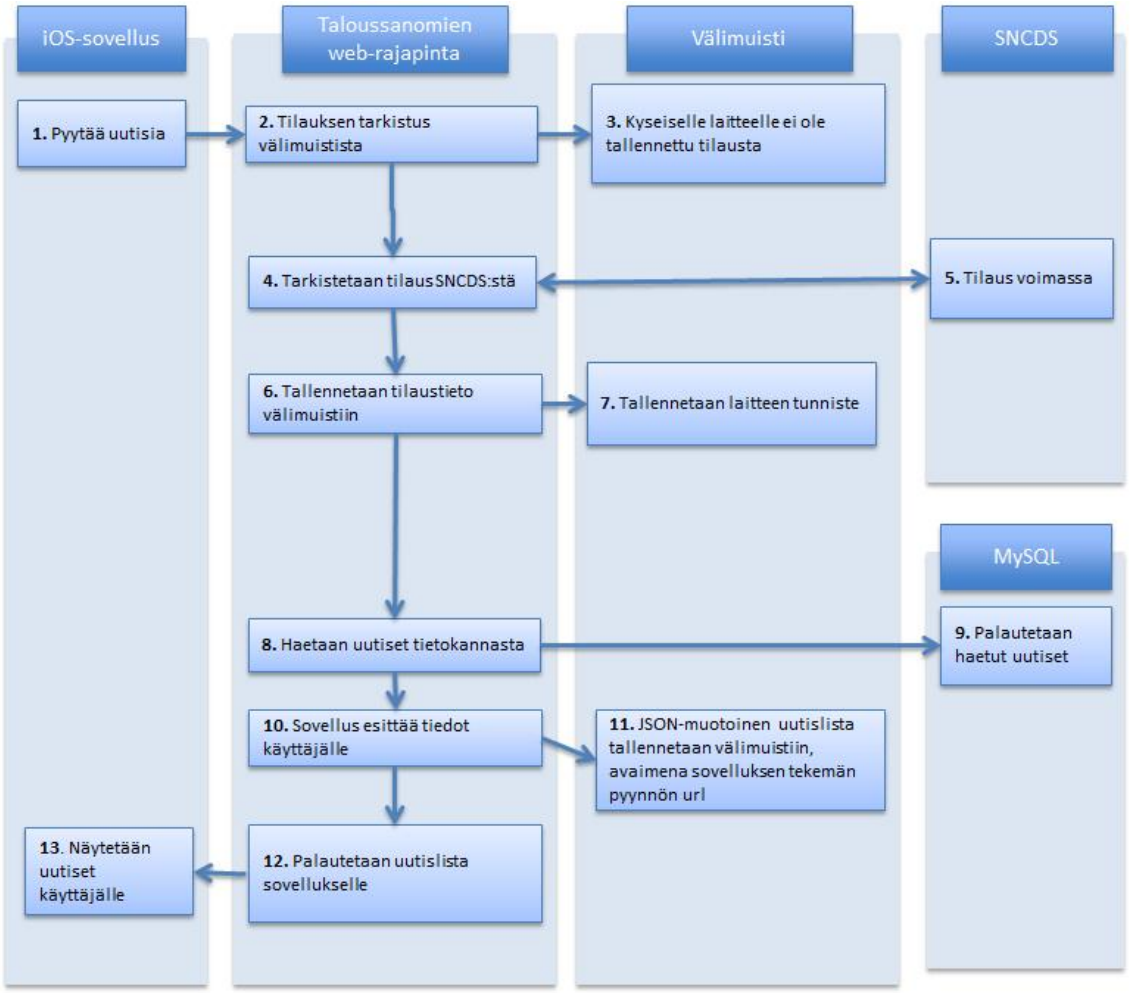
Taloussanomat+-palvelun tilauksessa on käytössä Auto-renewal-toiminto, eli tilaukset uusitaan automaattisesti tilausjakson päättyessä, ellei käyttäjä ole erikseen ilmoittanut, ettei halua uusia tilausta. Auto-renewal on AppStoren mahdollistama toiminto, eli SNCDS:ään tallennetut tilaukset päättyvät aina tilausjakson päättyessä ja sovellukselle palautetaan tieto, että tilaus on päättynyt. Tästä johtuen sovellus tarkistaa vielä erikseen AppStorelta, onko tilaus uusiutunut ja tallentaa sitten SNCDS:ään uuden tilauksen Taloussanomien rajapinnan kautta.

Koetilauksen teossa ei käytetä Applen rajapintaa, koska maksusuoritusta ei ole. Koetilaus-resurssille (/customer/save) annetaan parametrina laitteen tunniste ja käyttäjän sähköpostiosoite, jotka tallennetaan Taloussanomien MySQL-tietokantaan ja lisäksi tehdään koetilaus SNCDS:n rajapintaan. Koetilauksen jälkeen sovellus käyttäytyy samoin kuin varsinaisen tilaajan kanssa. Kun koetilaus on päättynyt, sovellus kutsuu tietyin väliajoin /customer/notify-resurssia, joka huomauttaa, jos käyttäjä on jo käyttänyt koejakson, mutta ei ole tilannut Taloussanomat+-palvelua.

5.3 Välimuistitus

Välimuistia käytetään sovelluksessa sekä asiakkaan että palvelimen päässä. Palvelin lähettää REST-periaatteita noudattaen http-viestin mukana tiedon, saako asiakas tallentaa viestin sisältöä välimuistiin. Tämän lisäksi palvelin ylläpitää omaa välimuistia, johon se tallentaa MySQL-tietokannasta ja SNCDS-taustajärjestelmästä noudetut tiedot, kuten JSON-muotoiset uutislistaukset ja käyttäjän tilausstatus.

Palvelimen välimuisti säilöö datan avain-arvo-pareina. Arvona käytetään kutsu-urlia parametreineen. Esimerkiksi uutishaun urlissa on mukana lueteltujen uutisten id:t. Välimuistiin tallennetaan siis erilaisia uutiskokoelmia. Jokaisesta uutiskokoelmasta tallennetaan kaksi eri versiota: tilaajan näkymä ja ei-tilaajan näkymä. Tilaajan näkymää tallennettaessa urlista poistetaan laitteen tunniste -parametri, jotta kokoelmia ei tallennettaisi käyttäjäkohtaisesti. Välimuistiin tallennetaan käyttäjäkohtaisesti ainoastaan käyttäjän tilausstatus.



Kuvio 4. Välimuistin käyttö uutishaussa.

Ennen jokaista uutishakua (tietokantakysely) ja tilauksen tarkistusta (SNCDS-kutsu) tarkistetaan, löytyykö välimuistista kutsu-urlin muotoista avainta. Jos avain löytyy, palautetaan sitä vastaava arvo välimuistista, muussa tapauksessa suoritetaan tietokantakysely tai SNCDS-kutsu, ja palautetaan niiden tuottama tulos. Tämä tulos tallennetaan välimuistiin, jotta seuraavalla kerralla, kun sovellus kutsuu samaa urlia, voidaan tulos hakea välimuistista.

5.4 Virheiden käsittely

Taloussanomien rajapinta kommunikoi kahden muun järjestelmän rajapinnan kanssa verkon ylitse. Sen lisäksi, että itse järjestelmissä voi tapahtua virhetilanteita tai niiden palvelimet ovat alhaalla, täytyy varautua myös verkkoyhteyksissä tapahtuviin häiriöihin.

Virhetilanteet on hoidettu hieman eri tavoin kutsusta ja tilanteesta riippuen. Jos tilauksen teko epäonnistuu, käyttäjälle näytetään virheilmoitus. Jos taas uutishaussa tilauksen tarkistus SNCDS:stä epäonnistuu, ladataan käyttäjälle karsittu (ei-tilaajan) uutislistaus. Jos uutishaussa myös tietokantakysely epäonnistuu, käyttäjälle näytetään virheilmoitus.

Virheen tapahtuessa palvelin lähettää asiakkaalle viestin, jonka statuksena on soveltuva http-virhekoodi. Lisäksi viestin sisältönä näytetään sovelluskohtainen virhekoodi ja tekstimuotoinen selitys virheestä. Rajapinnan sisäisessä virheen käsittelyssä käytetään apuna PHP:n sisäänrakennettua Exception-oliota. Kun joku rajapinnan olio havaitsee virheen, esimerkiksi Applen rajapintaan ei saada yhteyttä, se luo Exception-olion ja heittää sen eteenpäin kutsujalle eli controller-luokalle. Controller luo Exception-olion virhekoodin ja tekstiselityksen perusteella JSON-muotoisen virhe-olion palautettavaksi iOS-sovellukselle.

5.5 REST-rajoitteiden soveltaminen rajapinnassa

Kuten aiemmin todettiin, kaikki web-rajapinnat toteuttavat määritelmän mukaan asiakas-palvelin-rajoitetta, joten niin myös Taloussanomien web-rajapinta. Lisäksi Taloussanomien web-rajapinta on tilaton ja välittää asiakkaalle tietoa välimuistin käytöstä. Taloussanomien rajapinta käyttää yhdenmukaisia HTTP-metodeita oikein, mikä on perusedellytys kerroksittaisen järjestelmän rajoitteelle: järjestelmä voi hyödyntää HTTP-protokollan tunnistavia välikomponentteja.

Yllä kuvatut rajoitteet toteutuvat useimmissa web-rajapinnoissa. Tärkein erottava tekijä eri web-rajapintojen välillä on Yhdenmukaisen rajapinnan rajoitteen soveltaminen. Taloussanomien web-rajapinta sijoittuu yhdenmukaisen rajapinnan alirajotteiden toteutusastetta kuvaavassa RMM-mallissa tasolle kaksi: rajapinta hyödyntää URI-protokollaa ja HTTP-metodeja, mutta ei hypermediaa sovelluksen tilakoneena. Jotkin resurssien esitykset sisältävät linkkejä toisiin esityksiin (esimerkiksi uutisissa on linkkejä toisiin uutisiin), mutta pääsääntöisesti sovellus siirtyy tilasta toiseen rakentamalla resurssien kutsu-urlit itse ennalta sovittujen periaatteiden mukaisesti.

URI:n ja HTTP:n käyttötapa on myös hieman kyseenalainen REST-periaatteiden suhteen. Rajapinta määrittelee usean resurssin, mutta ne ovat lähinnä palvelutyyppisiä resursseja, joita voi kutsua ainoastaan ennalta määrätyllä HTTP-metodilla (vaikkakin kyseistä HTTP-metodia käytetään oikein). Vallitsevan käsityksen mukaan ollakseen todella REST-pohjainen, jokainen uutinen ja tilaus pitäisi määritellä omiksi resursseikseen. Taloussanomien rajapinta olisi silloin enemmän tämän näköinen (HTTP-pyyntöinä kuvattuna):

```
GET /news/{id}
GET /front_page/{pvm}/{osio}
GET /orders/{laitteen id}
PUT /orders/{laitteen id}?type=tyyppi
GET /notification/{laitteen id}
```

REST-pohjaisemman version voisi toteuttaa muillakin tavoilla, mutta tässä yksi. REST:n ideaali-url-rakennetta ei voi aivan täysin soveltaa nykyiseen sovellukseen: asiakkaan ja rajapinnan välisessä kommunikoinnissa olisi pitänyt huomioida RESTisyys jo suunnittelun alkuvaiheessa, jos olisi haluttu noudattaa tarkasti totuttua REST-pohjaista url-rakennetta.

Kuitenkin esimerkissä on oleellisia muutoksia aiempaan: Jokainen uutinen on erillinen resurssi (/news/{id}) samoin kuin tilaukset (/orders/{laitteen id}). Tilaukset eroavat uutisista oleellisesti resursseina siinä, että asiakas voi itse lisätä niitä järjestelmään PUT-metodilla. GET orders vastaisi siis alkuperäisen rajapinnan tilauksen voimassaolon tarkistus toimintoa ja PUT orders tilauksen tallennusta. GET /notification/ vastaa alkuperäistä notification metodia. GET /front_page/{pvm}/{osio} vastaa uutisruudukon hakua.

Yksi selkeä ongelma tässä versiossa on, että kokonaisia uutisia voi hakea vain joko yhden (GET /news/{id}) tai kaikki (GET /news/) kerrallaan. Suorituskyvyn parantamiseksi sovelluksen olisi voitava hakea tietty joukko uutisia kerrallaan. Uutisjoukko määräytyy sen perusteella, mitä uutista käyttäjä kulloinkin lukee; välimuistiin haetaan muutama seuraava ja edellinen uutinen. REST-pohjainen versio edellyttäisi erillistä kutsua jokaiselle uutiselle tai jonkinlaista palveluresurssia, joka palauttaisi annetun uutisen tunnisteen perusteella läheiset uutiset.

Enemmän REST-periaatteita noudattava rajapinta on helposti lähestyttävissä sellaiselle henkilölle, joka jo tuntee REST-periaatteet. Taloussanomien rajapinnan tapauksessa rajapintaa tulkitsivat kuitenkin ainoastaan Taloussanomien iOS-sovelluksen kehittäjät, jotka olivat myös mukana rajapinnan suunnittelussa. Rajapintaa ja sovellusta kehitettiin rinnakkain; jos jokin sovelluksen toiminnallisuus edellytti rajapinnalta jotain toimintoa, sellainen toteutettiin tai olemassa olevaa toimintoa muokattiin. Tällaisessa tilanteessa on tarpeetonta noudattaa tiukasti vallitsevia nimeämiskäytäntöjä. Kuitenkin tulevaisuutta ajatellen, jos Taloussanomien rajapinta avataan joskus julkiseen käyttöön, REST-käytäntöjä enemmän noudattava rajapinta voisi olla hyödyllinen.

HTTP:n ja URI:n käyttöä olisi voinut vielä harkita rajapintaa suunniteltaessa. Sen sijaan hypermedian käyttöä tilakoneena (viimeinen yhdenmukaisen rajapinnan rajoite) on hyvin vaikea ajatella tällaisen sovelluksen yhteydessä. Hypermedia vie yhdenmukaisen rajapinnan vielä astetta pidemmälle. Jos muista yhdenmukaisen rajapinnan piirteistä on voitu joustaa, koska rajapinnan käyttäjäryhmä on tarkkaan rajattu, ei hypermedian tuomalle lisäyhdenmukaisuudelle ole juuri perusteita.

Hypermedian oikeaoppinen käyttö edellyttää, että asiakkaan täytyy tuntea rajapinnasta ainoastaan yksi url, kuten tavallisen web-sivuston yhteydessä. Jatkotoimenpiteet löytyvät ensimmäisen resurssin palauttaman esityksen sisältämistä linkeistä. Esimerkiksi Taloussanomien web-rajapinta voisi koostua ainoastaan /frontpage-resurssista, joka palauttaisi koko sovelluksen etusivun. Etusivu sisältäisi samat toiminnot kuin käyttäjälle näkyvä etusivu, mutta koneen käsiteltäväksi tarkoitetussa muodossa. Sovellus päättäisi edelleen käyttöliittymän ulkoasun ja rakenteen, mutta palvelin vastaisi sovelluksen tilakoneesta, millaisiin tiloihin kustakin sovelluksen tilasta voisi siirtyä.

6 Yhteenveto

Tämän työn tarkoituksena oli perehtyä REST-arkkitehtuurityyliin ja sen sovellusmahdollisuuksiin web-rajapinnoissa. Työn lähtökohtana minulla oli hatara näkemys REST:stä jonkinlaisena uutena tapana toteuttaa web-rajapintoja. Olin kohdannut useita web-rajapintoja, joiden väitettiin olevan REST-rajapintoja, mutta jokaista tällaista väitettä vastaan tuntui olevan vastaväite, että tämä ei ainakaan ole

REST-rajapinta. Lisäksi itse web-rajapinnan käsitteestä tuntui olevan vaihtelevia käsityksiä, toisinaan samassa yhteydessä puhuttiin web serviceistä ja toisinaan sanottiin kaikkien web-sivujen olevan web serviceitä tai rajapintoja.

Lähdin siis liikkeelle REST:n alkulähteeltä, REST-raamattunakin pidetystä Roy Fieldingin väitöskirjasta. Aiheeseen perehtyessäni paljastui, että REST on paljon muutakin kuin vain yksi tapa toteuttaa web-rajapintoja: kyseessä on koko webin kantava arkkitehtuurityyli, joka oli mukana webin kehityksessä jo sen alkuajoilta lähtien. Koko maailmanlaajuista webiä kehittämässä on kuitenkin ollut maailmanlaajuinen joukko arkkitehtejä omine näkemyksineen ja tavoitteineen, jonka seurauksena web-sovelluksia on kehitetty vähän milloin mitenkin. Erityisesti tämä on näkynyt koneiden välisiä sovelluksia kehitettäessä, jolloin etäyhteyden takana olevien komponenttien kutsuminen on pyritty abstraktoimaan lokaaliksi kutsuksi välittämättä lainkaan etäkommunikointiin liittyvistä haasteista.

Web on maailmanlaajuinen kokoelma HTTP-protokollan avulla keskustelevia komponentteja. Tämän työn puitteissa määrittelin web-rajapinnan sellaiseksi (webissä sijaitsevaksi) rajapinnaksi, jota käytetään nimenomaan koneiden välisessä kommunikoinnissa erotuksena ihmisrajapinnoista, joita käytetään web-selaimen ja koneen välisessä kommunikoinnissa. Tämän erottelun tehtyäni totesin, että on olemassa karkeasti ottaen kaksi tapaa toteuttaa web-rajapintoja: REST-pohjaiset web-rajapinnat, jotka ottavat huomioon etäyhteyksien tuomat haasteet ja RPC-pohjaiset web-rajapinnat, jotka pyrkivät lakaisemaan etäyhteyksien tuomat hankaluudet maton alle.

REST-arkkitehtuurityyli on erityisen hyödyllinen yksinkertaisissa, mutta laajalle yleisölle tarkoitetuissa web-rajapinnoissa, jotka edellyttävät hyvää suorituskykyä ja toimintavarmuutta. RPC-pohjainen SOAP-protokolla tarjoaa tapoja toteuttaa joitain monimutkaisempia toimintoja, kuten transaktioita, mutta samalla se lisää huomattavasti verkon kuormitusta ja aiheuttaa monimutkaisuudessaan kauhunväristyksiä monelle web-kehittäjälle.

Lopuksi esittelin Taloussanomien iOS-sovelluksen käyttöön kehittämää web-rajapintaa, joka sijoittuu jonnekin REST- ja RPC-pohjaisten rajapintojen välimaastoon, kuten useimmat web-rajapinnat. REST-periaatteiden tiukasta noudattamisesta voitiin luistaa

sillä web-rajapinta oli tarkoitettu ainoastaan iOS-sovelluksen kehitystiimin käyttöön, jolloin maailmanlaajuisen webin kanssa täysin yhdenmukainen rajapinta ei ollut tärkeysjärjestyksessä korkeimpana.

Lähteet

Fielding, Roy Thomas, Architectural Styles and the Design of Network-based Software Architectures [väitöskirja]. Kalifornian yliopisto, Irvine. 2000.

RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, R. Fielding, J. Gettys, J. Mogul, H. Frystuk, L. Masinter, P. Leach, T. Berners-Lee. 1999. Verkkodokumentti IETF. <http://www.ietf.org/rfc/rfc2616>. Luettu 31.3.2012.

RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter. 1998. Verkkodokumentti. IETF. <http://www.ietf.org/rfc/rfc2396>. Luettu 31.3.2012.

Rodriguez, Alex, RESTful Web services: The basics. 2008. Verkkodokumentti. IBM. <www.ibm.com/developerworks/webservices/library/ws-restful/>. Päivitetty 6.11.2008. Luettu 25.3.2012.

TCP/IP model. Verkkodokumentti. Wikipedia. 2012. <http://en.wikipedia.org/wiki/TCP/IP_model>. Luettu 31.3.2012.

Internet media type. Verkkodokumentti. Wikipedia. 2012. <http://en.wikipedia.org/wiki/Internet_media_type>. Luettu 1.4.2012.

Tilkov, Stefan, A Brief Introduction to REST. 2007. Verkkodokumentti. InfoQueue. <http://www.infoq.com/articles/rest-introduction>. Luettu 7.4.2012.

CRUD. Verkkodokumentti. Wikipedia. 2012 <http://en.wikipedia.org/wiki/Create,_read,_update_and_delete>. Luettu 1.4.2012.

Richardson, Leonard - Ruby, Sam, RESTful web service. 2007. Verkkokirja. O' Reilly Media. <http://books.google.com/books?id=XUaErakHsoAC&printsec=frontcover#v=onepage&q&f=false>. Luettu 9.4.2012.

Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, Dave Winer, Simple Object Access Protocol (SOAP) 1.1. 2000. Verkkodokumentti. W3C. <http://www.w3.org/TR/2000/NOTE-SOAP-20000508 >. Luettu 13.4.2012.

Spies, Brennan, Web Services, Part 1: SOAP vs. REST. 2008. Verkkodokumentti. <http://www.ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest>. Luettu 13.4.2012.

XML-RPC. Verkkodokumentti. Wikipedia. 2012 <http://en.wikipedia.org/wiki/XML-RPC>. Luettu 13.4.2012.

Vinoski, Steve, RPC and its Offspring: Convenient, Yet Fundamentally Flawed. 2009. Video. <http://www.infoq.com/presentations/vinoski-rpc-convenient-but-flawed>. Katsottu 14.4.2012.

Box, Don, Don Box Discusses SOAP, XML, REST and M. 2010. Video. <http://www.infoq.com/interviews/box-soap-xml-rest-m>. Katsottu 14.4.2012.

Rozlog, Mike, REST and SOAP: When Should I Use Each (or Both)?. 2010. Verkkodokumentti. InfoQueue. <http://www.infoq.com/articles/rest-soap-when-to-use-each>. Luettu 15.4.2012.

Richardson, Leonard, Act Three: The Maturity Heuristic. 2008. Verkkodokumentti. <http://www.crummy.com/writing/speaking/2008-QCon/act3.html >. Luettu 15.4.2012.